Лекція 13

Списки, кортежі та множини

Записати результат роботи програми (якщо Ви вважаєте, що результатом є помилка, напишіть – «помилка» або «error»):

1	x = ["2016","2017"]; y = x.copy() y[1] = "2018" ; print(x,y)	15	r = range(0, 36, 6) for i in r: print(i, end = " ")
2	s = ("я", "str", "рядок", 10, 42.1, False) b = list(s[4:1:-1]); print(b)	16	arr = [[] for i in range(2)] arr[0].append(3); print(arr)
3	arr = []; arr.append(1) arr.append("byte"); print(arr)	17	arr = [[] for i in range(2)] arr[1].append(4); print(arr)
4	x, y = [2, 3], [2, 3] ; x[1] = 50; print(x, y)	18	x = y = [2, 3]; x[1] = 50; print(x, y)
5	>>> x, y = [1, 2], [1, 2] ; x is y	19	>>> x = [1, 2, 3, 4, 5]; z = x[:]; z is x
6	x = [1, [4, 5]]; y = list(x); y[0] = 200 y[1][1] = 100; print(x,y)	20	import copy; x = [1, [4, 5]] y = copy.deepcopy(x); y[1][1]=100; print(x,y)
7	>>> nested = [[1, 2, 3],[4, 5, 6],[7, 8, 9]] >>> nested[0]; nested[1][2]	21	import copy; x = [1, 2]; y = [x, x] z = copy.deepcopy(y); z [0][0]=300; print(y,z)
8	>>> arr = [1, "str", 4.1, "5"]; arr[1] ; arr[-3]	22	>>> x, y, *z = [2, 10, 3, 9, 5]; x, y, z
9	>>> x, *y, z = [2, 4, 6, 8, 10]; x, y, z	23	>>> x, *y, *z = [1, 2, 3, 4, 5]; x, y, z
10	>>> arr = [1, 2, 3, 4, 5, 6]; arr[6]	24	>>> arr=[1, 2, 3, 4]; arr[-4]=100; arr[-4], arr[-1]
11	>>>arr=[3, 4, 5, 6];arr[3]=100;arr[-4]=50;arr	25	>>> arr = [1, 2, 3, 4, 5, 6]; f = arr[1:6:2]; f
12	>>> arr = [5, 1, 7, 2, 8, 3]; arr[2:]=arr[4:]*2; arr	26	>>> arr = [5, 10, 15, 20, 25, 30]; arr[1:]
13	>>> arr = [5, 10, 15, 20, 25, 30]; arr[:-1]	27	>>> arr = [5, 10, 15, 20, 25, 30]; arr[::-1]
14	>>> arr = [5, 10, 15, 20, 25, 30];arr[-1:]	28	>>> arr = [5, 1, 5, 2, 2, 3];arr[-1:]=[8]; arr
		29	>>> arr = [5, 7, 9, 1, 2, 3];arr[1:3] = [8,13];arr

Пошук елемента в списку

```
A = [0,3,5,7,10,20,28,30,45,56]
```

X-змінна для пошуку

```
i=0 ; m=int(j/2) ;  j=len(A)-1
```

Якщо `A[m]<X`–шукаємо справа від `m:` `i=m+1 ; j=len(A)-1`

```
[28,30,45,56]
```

Якщо `A[m]>X`–шукаємо зліва від `m:` `i=0; j=m-1`

```
[0,3,5,7,10]
```

Якщо `A[m]=X`–пошук завершено .

Бінарний пошук

```
A = [0,3,5,7,10,20,28,30,45,56]
x = 45
i = 0
j = len(A)-1
m = int(j/2)
while A[m] != x and i < j:
    if x > A[m]:
        i = m+1
    else:
        j = m-1
    m = int((i+j)/2)

if i > j:
    print('Елемент не знайдено')
else:
    print('Індекс елемента: ', m)
```

Перевірка факту входження елемента

Перевірка на *входження* елемента в список

Оператор `in:` якщо елемент входить у список, то повертається значення `True`, а якщо ні, то – `False`.

Перевірка на *невходження* елемента в список

Оператор `not in` виконує перевірку на *невходження* елемента в список: якщо елемент відсутній у списку, повертається `True`, а якщо ні, то – `False`.

Приклад 1.

```
# Перевірка на входження
>>> 2 in [1, 2, 3, 4, 5], 6 in [1, 2, 3, 4, 5]
(True, False)
# Перевірка на невходження
>>> 2 not in [1,2,3,4,5], 6 not in [1,2,3,4,5]
(False, True)
```

Метод `index()`

Щоб довідатися індекс елемента всередині списку, слід скористатися методом `index()`. Формат методу:

```
index(<Значення>[, <Початок>[, <Кінець>]])
```

1. Метод `index()` повертає індекс елемента, що має зазначене значення.

2. Якщо значення не входить у список, то виконується виключення `Valueerror`.

3. Якщо другий і третій параметри не зазначені, то пошук буде проводитися з початку й до кінця списку.

Приклад застосування методу `index()`

Приклад 2.

```
>>> arr = [1, 2, 1, 2, 1, 7]

>>> arr.index(1), arr.index(2)
(0, 1)

>>> arr.index(1, 1), arr.index(1, 3, 5)
(2, 4)

>>> arr.index(3)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Valueerror: 3 is not in list
```

Метод `count()`

Довідатися загальну кількість елементів з зазначеним значенням дозволяє метод `count(<Значення>)`

1. Якщо елемент не входить у список, то повертається значення 0.

Приклад 2.

```
>>> arr = [1, 2, 1, 2, 1]

>>> arr.count(1 ), arr.count(2)
(3, 2)

>>> arr.count(3)# Елемент не входить у список
0
```

Пошук мінімального елемента

Алгоритм пошуку мінімуму

```
s = [2,4,1,3]
m = 0
i = 1
while i < len(s):
    if s[i] < s[m]:
        m = i
    i += 1
print (s[m])
```

Алгоритм пошуку максимуму

```
s = [2,4,1,3]
m = 0
i = 1
while i < len(s):
    if s[i] > s[m]:
        m = i
    i += 1
print (s[m])
```

Функції `max()` і `min()`

За допомогою функцій `max()` і `min()` можна довідатися максимальне й мінімальне значення списку відповідно.

Приклад 3.

```
>>> arr = [1, 2, 3, 4, 5, 6]
>>> max(arr), min(arr)
(6, 1)

>>> arr = ["a", "b", "c", "d", "e", "f"]
>>> min(arr), max(arr)
('a', 'f')

>>> arr = ["a", "aa", "aaa", "aaaa"]
>>> min(arr), max(arr)
('a', 'aaaa')
```

Функція any()

1. Функція any (<Послідовність>) повертає значення True, якщо в послідовності існує хоч один елемент, який у логічному контексті повертає значення True.
2. Якщо послідовність не містить елементів, повертається значення False.

Приклад 4.

```
>>> any([0, None])
False
>>> any([])
False
>>> any([0, None, 1])
True
>>> any(["a"])
True
```

Функція `all`

1. Функція `all (<Послідовність>)` повертає значення `True`, якщо всі елементи послідовності в логічному контексті повертають значення `True` або послідовність не містить елементів.

Приклад 5.

```
>>> all([18, "Petrenko"])
True
>>> all([])
True
>>> all([0, "Petrenko"])
False
>>> all([18, ""])
False
>>> all([0, None])
False
```

Перевертання й перемішування списку

Метод `reverse()`

Метод `reverse()` змінює порядок проходження елементів списку на протилежний.
Метод змінює поточний список і нічого не повертає.

Приклад 6.

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr.reverse() # Змінюється поточний список
>>> arr
[5, 4, 3, 2, 1]

>>> arr = ["a", "b", "c", "d"]
>>> arr.reverse()
>>> arr
['d', 'c', 'b', 'a']
```

Функція `reversed`

1. Подібність із `reverse`

Функція, як і метод `reverse`, змінює порядок проходження елементів списку на протилежний.

2. Відмінність від `reverse`

Дає можливість одержати новий список зі зворотним порядком.

Формат функції:

```
reversed(<Послідовність>).
```

1. Функція повертає ітератор

2. Список можна одержати за допомогою функції `list()` або генератора списків

Приклад 7. Застосування функції reversed

```
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> reversed(arr)
<list_reverseiterator object at 0x02E8A530>

>>> list(reversed(arr)) # Використання list()
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

# Вивід за допомогою циклу
>>> for i in reversed(arr): print(i, end=" ")
10 9 8 7 6 5 4 3 2 1

# Використання генератора списків
>>> [i for i in reversed(arr)]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Функція перемішування shuffle

Формат функції:

shuffle(<Список>])

1. **Функція з модуля** random
2. «Перемішує» список випадковим чином. Функція перемішує список і нічого не повертає.

Приклад 8.

```
>>> import random # Підключаємо модуль random
>>> arr = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Перемішуємо список випадковим чином
>>> random.shuffle(arr)
>>> arr
[9, 5, 7, 8, 1, 3, 6, 2, 10, 4]
```

Вибір елементів випадковим чином

Одержати елементи зі списку випадковим чином дозволяють функції з модуля `random`.

Функція `choice`

`choice (<Послідовність>)`

Функція повертає випадковий елемент із будь-якої послідовності (рядка, списку, кортежу):

```
>>> import random #підключаємо модуль random
# Список
>>>random.choice(["s","t", "r", "i", "n", "g"])
't'
# Рядок
>>> random.choice("programming")
'i'
# Кортеж
>>> random.choice((1, 1.12345, "tuple"))
1.12345
```

Функція `sample`

`sample(<Послідовність>, <Кількість елементів>)`

1. Повертає список із зазначеної кількості елементів.
2. У цей список потраплять елементи з послідовності, обрані випадковим чином.
3. Як послідовність можна вказати будь-які об'єкти, що підтримують ітерації.

Приклад 9.

```
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random.sample(arr, 2)
[3, 7]
>>> random.sample(arr, 4)
[4, 8, 10, 2]
>>> arr # Сам список не змінюється
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Сортування

Сортування вибором

```
a=[7, 3, 4,  5, 9]
for i in range(len(a)-1):
    for  j in range(i+1, len(a)-1):
        if  a[i]>a[j]:
            a[i], a[j] = a[j], a[i]
print(a)
```

Бульбашкове сортування

```
a = [5,2,7,4,0,9,8,6]
n = 1
while n < len(a):
     for i in range(len(a)-n):
          if a[i] > a[i+1]:
               a[i],a[i+1] = a[i+1],a[i]
     n += 1
print(a)
```

Сортування списку

Відсортувати список дозволяє метод `sort()`.
Метод має наступний формат:

```
sort([key=None][, reverse= False])
```

1. Усі параметри не є обов'язковими.
2. Метод змінює поточний список і нічого не повертає.
3. Параметр `key` може вказувати на функцію, що задає умови сортування

Приклад сортування за зростанням
(параметр `reverse= False` за замовчуванням)

Приклад 10.

```
>>> arr = [2, 7, 10, 4, 6, 8, 9, 3, 1, 5]
>>> arr.sort()# Змінює поточний список
>>> arr
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Приклад сортування за спаданням

(параметр `reverse= True`)

Щоб відсортувати список за спаданням, слід в параметрі `reverse` указати значення `True`:

Приклад 11.

```
>>> arr = [7, 2, 10, 4, 8, 6, 9, 3, 1, 5]
>>> arr.sort(reverse = True)
>>> arr # Сортування за спаданням
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> arr = ["sky","land","water","fire","sun"]
>>> arr.sort(reverse = True)
>>> arr
['water', 'sun', 'sky', 'land', 'fire']

>>> arr = ["sky","land","water","fire","Sun"]
>>> arr.sort(reverse = True)
>>> arr
['water', 'sky', 'land', 'fire', 'Sun']
```

Сортування залежить від регістру

Стандартне сортування залежить від регістру символів

Приклад 12.

```
arr = ["ant", "Asia", "bee", "Brazil"]
arr.sort()
for i in arr:
    print(i, end=" ")
# Результат виконання: Asia Brazil ant bee
```

Щоб регістр символів не враховувався, можна вказати посилання на функцію для зміни регістру символів у параметрі `key`

Приклад 13.

```
arr = ["ant", "Asia", "bee", "Brazil"]
arr.sort(key=str.lower)# Указуємо метод lower()
for i in arr:
    print(i, end=" ")
# Результат виконання: ant Asia bee Brazil
```

Інші застосування параметра `key`

1. У параметрі `key` можна вказати функцію, що виконує будь-яку дію над кожним елементом списку.
2. Як єдиний параметр вона повинна приймати значення чергового елемента списку, а як результат – повертати результат дій над ним.
3. Цей результат буде брати участь у процесі сортування, але значення самих елементів списку не зміняться.

Приклад 14. Сортування по першому елементу

```
>>> def getkey(item):
        return item[0]
>>> s = [[10, 3], [1, 7], [9, 34], [3, 64]]
>>> s.sort(key=getkey);s
[[1, 7], [3, 64], [9, 34], [10, 3]]
```

4. Метод **`sort()`** сортує сам список і не повертає жодного значення.

Функція `sorted()`

Функція `sorted()` формує новий список, а поточний список залишає без змін.

`sorted(<Послідовність>[, key=None] [, reverse= False])`

1. Перший параметр `<Послідовність>` повинен містити список, який необхідно відсортувати.

2. Параметр `key` може вказувати на функцію, що задає умови сортування

3. Параметр `[reverse= False]` сортувати за зростанням.
Параметр `[reverse= True]` сортувати за спаданням.

Приклад 15. Застосування функції `sorted()`

```
>>> arr = [7, 10, 4, 2, 6, 8, 9, 3, 1, 5]

>>> sorted(arr) # Повертає новий список!
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> sorted(arr, reverse=True) # Повертає новий
список!
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

arr = ["Asia","bee", "ant", "Brazil"]
arr1 = sorted(arr,key=str.lower) # метод lower()
for i in arr1:
    print(i, end=" ")
# Результат виконання: ant Asia bee Brazil
```

Заповнення списку числами

Створити список, що містить діапазон чисел, можна за допомогою функції `range()`. Функція повертає діапазон, який перетворюється на список викликом функції `list()` .
Функція `range()` має наступний формат:

```
range([<Початок>, ] <Кінець> [, <Крок>] )
```

1. Перший параметр `<Початок>` задає початкове значення – якщо він не зазначений, використовується значення 0.
2. У другому параметрі `<Кінець>` вказується кінцеве значення. Кінцеве значення не входить у діапазон, що повертається.
3. Якщо параметр `<Крок>` не зазначений, то використовується значення 1.

Приклади застосування функції `range()`

Приклад заповнення списку числами від 0 до 10:

Приклад 16.

```
>>> list(range(11))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Створимо список, що містить діапазон чисел від 1 до 15:

Приклад 17.

```
>>> list(range(1,16))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15]
```

Тепер змінимо порядок проходження чисел на протилежний:

```
>>> list(range(15, 0, -1))
[15,14,13,12,11,10,9,8,7,6,5,4,3,2,1]
```

Перетворення списку в рядок. Метод `join()`

Перетворити список у рядок дозволяє метод `join()`. Елементи додаються через зазначений роздільник.

Формат методу:

```
<Рядок>=<Роздільник>.join(<Послідовність>)
```

Приклад 18.

```
>>> arr = ["word1", "word2", "word3"]
>>>"-".join(arr)
'word1 - word2 - word3'
```

Елементи списку повинні бути рядками, інакше повертається виключення `Typeerror`: **Приклад 19.**

```
>>> arr = [ "word1", "word2", "word3", 2]
>>> " - ".join(arr)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Typeerror: sequence item 3: expected str instance,
int found
```

Як уникнути виключення в методі `join()`

Уникнути цього виключення можна за допомогою виразу-генератора, усередині якого поточний елемент списку перетворюється в рядок за допомогою функції `str()`:

Приклад 20.

```
>>> arr = ["word1", "word2", "word3", 2]
>>> " - ".join(( str(i) for i in arr ))
'word1-word2-word3-2'
```

Крім того, за допомогою функції **`str()`** можна відразу одержати строкове представлення списку:

Приклад 21.

```
>>> arr = [ "word1", "word2", "word3", 2]
>>> str(arr)
"['word1', 'word2', 'word3', 2]"
```

Кортежі

Подібність зі списками – є впорядкованими послідовностями елементів.
Відмінність від списків – змінити кортеж не можна.

Можна сказати, що кортеж – це список, доступний тільки для читання.

Створити кортеж можна за допомогою функції

```
tuple([<Послідовність>])
```

1. Функція дозволяє перетворити будь-яку послідовність у кортеж.
2. Якщо параметр <Послідовність> не зазначений, то створюється порожній кортеж.

Приклад 22. Приклади створення кортежу з **tuple**

```
>>> tuple()# Створюємо порожній кортеж
()


>>> tuple("String") # Перетворимо рядок у кортеж
 ('S', 't', 'r', 'i', 'n', 'g' )


>>> tuple([1, 2, 3, 4, 5]) # Перетворимо список у
кортеж
(1, 2, 3, 4, 5)
```

Створення кортежу безпосереднім задаванням елементів

Кортеж задають, указавши всі елементи через кому усередині круглих дужок (або без дужок):

Приклад 23.

```
>>> t1 = ()    # Створюємо порожній кортеж
>>> t2 = (5,) # Створюємо кортеж з одного елемента
>>>  t3  =(1,  "str",  (3,  4))  #  Кортеж  із  трьох
елементів
>>>  t4  =  1,  "str",  (3,  4)    #  Кортеж  із  трьох
елементів без дужок
>>> t1, t2, t3, t4
((), (5,), (1, 'str', (3, 4)), (1, 'str', (3, 4)))
```

Для створення кортежу необхідна кома

1. Щоб створити кортеж з одного елемента, необхідно наприкінці вказати кому

```
>>> t = (5,).
```

Саме коми формують кортеж, а не круглі дужки. Якщо усередині круглих дужок немає ком, то буде створений об'єкт іншого типу.

Приклад 24.

```
>>> t =(5); type(t)
<class 'int'>
# Одержали число, а не кортеж!
>>> t = ("str"); type(t) # Одержали рядок, а не кортеж!
<class 'str'>
```

Не дужки формують кортеж, а коми.

Будь-який вираз в мові Python можна взяти в круглі дужки.

Структура кортежів

1. Позицію елемента в кортежі задають *індексом.*

2. Нумерація елементів кортежу (як і списку) починається з 0.

3. Як і всі послідовності, кортежі підтримують:

- доступ до елемента по індексу [],
- одержання зрізу [::],
- конкатенацію (оператор +),
- повторення (оператор *),
- перевірку на входження (оператор **`in`**)
- перевірку на невходження (оператор **`not in`**).

Приклад 25

```
>>> t = (1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> t[0]  # Одержуємо значення першого елемента кортежу
1
>>> t[::-1]        # Змінюємо порядок проходження елементів
(9, 8, 7, 6, 5, 4, 3, 2, 1)
>>> t[2:5]              # Одержуємо зріз
(3, 4, 5)
>>> 8 in t, 0 in t      # Перевірка на входження
(True, False)
>>> ( 1, 2, 3)*3
 (1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> (1, 2, 3) + (4, 5, 6) # Конкатенація
(1, 2, 3, 4, 5, 6)
```

Кортежі - незмінювані типи даних

Кортежі, як уже неодноразово відзначалося, є незмінюваними типами даних. Іншими словами, можна одержати елемент по індексу, але змінити его не можна:

Приклад 26

```
>>> t = (1, 2, 3) # Створюємо кортеж
>>> t[0] # Одержуємо елемент по індексу
1

>>> t[0] = 50 # Змінити елемент по індексу не можна!
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Typeerror: 'tuple' object does not support item Assignment
```

Функції й методи для кортежів

Кортежі підтримують уже знайомі нам по списках функції `len()`, `min()`, `max()`, методи `index()` і `count()`.

Приклад 27.

```
>>> t = (1, 2, 3) # Створюємо кортеж

>>> len(t) # Одержуємо кількість елементів
3

>>> t = ( 1, 2, 1, 2, 1)

# Шукаємо елементи в кортежі
>>> t.index(1), t.index(2)
(0, 1)
```

Множини

Множина – це неупорядкована послідовність унікальних елементів, з якою можна порівнювати інші елементи, щоб визначити, чи належать вони цій множині.

Оголосити множину можна за допомогою функції `set()`:

Приклад 28.

```
>>> s = set()
>>> s
set()
```

Функція `set()` дозволяє також перетворити елементи послідовності в множину:

Приклад 29. Перетворення в множину

```
>>> set("string") # Перетворимо рядок
{'i', 'r', 'g', 's', 'n', 't'}

>>> set([1, 2, 3, 4, 5]) # Перетворимо список
{1, 2, 3, 4, 5}

>>> set((1, 2, 3, 4, 5)) # Перетворимо кортеж
{1, 2, 3, 4, 5}

>>> set([1, 2, 3, 1, 2, 3]) #залишаються тільки
унікальні
{1, 2, 3}
```

Інші способи перетворення в множину

Перебрати елементи множини дозволяє цикл `for`:

Приклад 30.

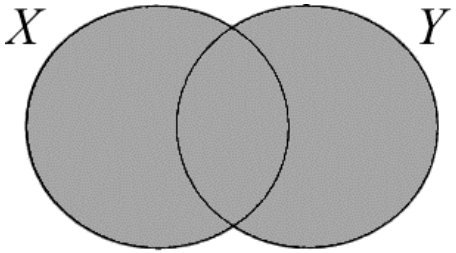
```
>>> for i in set([1, 2, 3]): print(i)
1 2 3
>>> for i in {1, 2, 3}: print(i)
1 2 3
```

Одержати кількість елементів множини дозволяє функція `len()`:

Приклад 31.

```
>>> len(set([1, 2, 3]))
3
```

Оператори й методи для роботи з множинами



Оператори **|** і **union()** – поєднують дві множини:

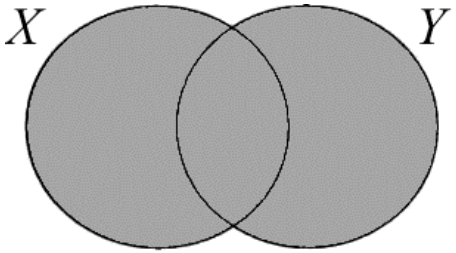
Приклад 32.

```
>>> s = set([ 1, 2, 3])
>>> s.union(set([4, 5, 6])), s | set([4, 5, 6])
({1, 2, 3, 4, 5, 6}, {1, 2, 3, 4, 5, 6})
```

Якщо елемент уже міститься в множині, то він повторно доданий не буде:

Приклад 33.

```
>>> set([1, 2, 3]) | set([1, 2, 3])
{1, 2, 3}
```



Оператори **`a |= b`** і **`a.update(b)`** – додають елементи множини `b` у множину `a`:

Приклад 35.

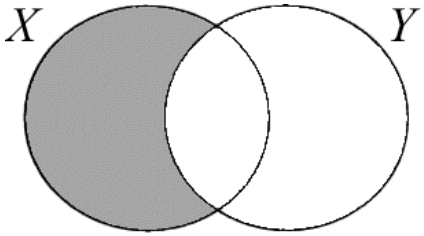
```
>>> s = set([1, 2, 3])

>>> s.update(set([4, 5, 6]))

>>> s
{1, 2, 3, 4, 5, 6}

>>> s |= set([7, 8, 9]); s
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Оператори **-** і **difference()** – обчислює різницю множин:



Приклад 36.

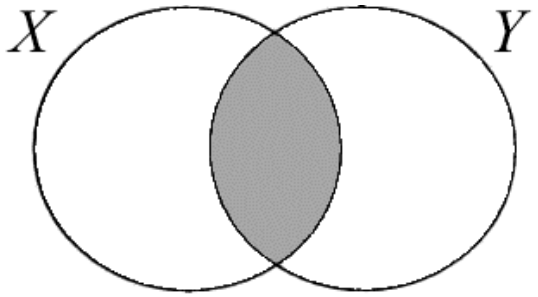
```
>>> set([1, 2, 3]) - set([1, 2, 4])
set([3])
>>> s = set([1, 2, 3])
>>> s.difference(set([1, 2, 4]))
set ([3])
```

Оператори **a -= b** і **a.difference_update(b)** – видаляють елементи із множини `a`, які існують і в множині `a`, і в множині `b`:

Приклад 37.

```
>>> s = set([1, 2, 3])
>>> s.difference_update(set([1, 2, 4])); s
{3}
>>> s -= set([3, 4, 5]); s
set()
```

Оператори **&** і **intersection()** – виконують перетин множин.



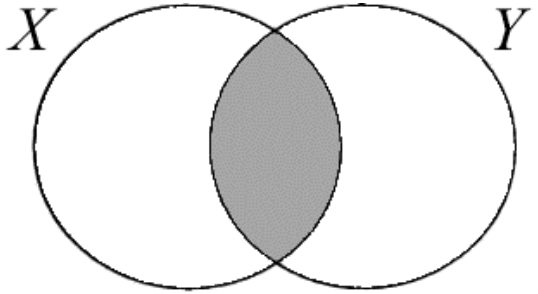
Дозволяють одержати елементи, які існують в обох множинах:

Приклад 38.

```
>>> set([1, 2, 3]) & set([1, 2, 4])
{1, 2}

>>> s = set([ 1, 2, 3])

>>> s.intersection(set([1, 2, 4]))
{1, 2}
```



Оператори **a &= b** і **a.intersection_update b** – у множині a залишаться елементи, які існують і в множині a й у множині b:

Приклад 39.

```
>>> s.intersection_update(set([1, 2, 4]))
>>> s
{1, 2}

>>> s &= set([1, 6, 7])
>>> s
{1}
```