Контрольні запитання

1. Що таке множина?

2. Як зберігається множина в пам'яті ЕОМ? Який максимальний обсяг оперативної пам'яті можна відвести під зберігання однієї множини?

3. Які операції можна виконувати над множинами?

4. Як додати елемент в множину?

5. Як видалити елемент з множини?

6. Як вивести елементи множини? Як підрахувати кількість елементів у множині?

Аудиторна робота

Пример 9.1: перетин послідовностей у вигляді рядків.

Пошук перетину двох послідовностей у вигляді рядків.

1. Розглянемо цикл *for*, який вибирає елементи, загальні для двох рядків. Після того як цикл *for* виконано, змінна *res* буде посилатися на список, який містить всі однакові елементи, виявлені в *seq1* і *seq2*:

```
seq1 = "spam"; seq2 = "scam"
res = [] # спочатку список пустий
for x in seq1: # виконати обхід першої послідовності
... if x in seq2: # загальний елемент?
... res.append(x) # Додати в кінець результата
res
```

2. Генератор списку

```
s1 = "spam"; s2 = "scam"
[x for x in s1 if x in s2]
```

Результат:

```
['s', 'a', 'm']
```

3. Використаємо множини для визначення перетину послідовностей у вигляді рядків

```
x=set("spam"); y = set("scam")
z=x & y # Перетин множин
print(z)
```

Результат

```
{'m', 'a', 's'}
```

Пример 9.2. Множини.

1. Створення множин – екземплярів класу *set*

```
my_set={1, 5, 3, 9}  # множина цілих чисел
print(my_set)
basket={'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket)
```

2. Створення множин на основі генераторів множин

```
number_set={i + j for i in range(10) for j in range(5)}
print(number_set)
char_set={x for x in 'abracadabra' if x not in 'abc'}
print(char_set)
```

3. Використання конструктору *frozenset*. Множини можуть включати об'єкти тільки незмінних типів. Якщо необхідно зберегти одну множину всередині іншої, створюють множину за допомогою вбудованої функції *frozenset*, яка діє так само, як функція *set*, але формує *незмінну множину*.

```
empty_set=set(); print(empty_set) # Пуста множина
empty_frozenset=frozenset() # Пуста незмінювана множина
print(empty_frozenset); my_frozenset=frozenset([4,1,3,8])
my_set=set(my_frozenset); print(my_set); print(my_frozenset)
```

4. Операції з множинами (*set, frozenset*)

```
my_set = {4, 5, 1, 2}
# Кількість елементів множини
print('len({}) = {}'.format(my_set, len(my_set)))
print()
# Перевірка входження елемента
print(4 in my_set); print(3 not in my_set)
print(9 in my_set);print()
# Чи перетинаються множини
print({3, 4, 5}.isdisjoint({8, 1, 0}))
print({3, 4, 5}.isdisjoint({1, 2, 3}));print()
# Перевірка включення однієї множини в іншу
print({1, 7, 9}.issubset({1, 2, 3, 7, 9}))
print({1, 7, 9} <= {1, 2, 3, 7, 9})
print({1, 7, 9, 2, 3} <= {1, 2, 3, 7, 9});print()
# Перевірка чіткого включення
print({1, 7, 9} < {1, 2, 3, 7, 9})
print({1, 7, 9, 2, 3} < {1, 2, 3, 7, 9});print()
# Перевірка включення однієї множини в іншу
print({1, 2, 3, 4}.issuperset({1, 2}))
print({1, 2, 4, 4} >= {1, 2})
print({1, 2, 3, 4} >= {1, 2, 3, 4});print()
# Перевірка чіткого включення
print({1, 2, 4, 4} > {1, 2})
print({1, 2, 3, 4} > {1, 2, 3, 4});print()
# Об'єднання множин
print({1, 3}.union({2, 3, 4}))
print({1, 3} | {2, 3, 4});print()
# Перетин множин
print({1, 3}.intersection({2, 3, 4}))
print({1, 3} & {2, 3, 4});print()
# Різниця множин
print({1, 2, 3, 4}.difference({3, 4, 5}))
print({1, 2, 3, 4} - {3, 4, 5});print()
# Симетрична різниця
print({1, 2, 3, 4}.symmetric_difference({3, 4, 5, 6}))
print({1, 2, 3, 4} ^ {3, 4, 5, 6});print()
# Копіювання множини
my_set = set('chars')
copy = my_set.copy();print(copy)
```

5. Операції над множинами, які є методами, мають в якості аргументів будь-які

ітерабельні об'єкти. Операції над множинами, записані у вигляді бінарних операцій, вимагають, щоб другий операнд операції теж був множиною, і повертають множину того типу, якою була перше множина

```
print(frozenset('abc').union(frozenset('cdef')))  # коректно
print(frozenset('abc') | frozenset('cdef'))  # коректно
print(frozenset('abc').union('cdef'))  # коректно
print(frozenset('abc') | 'cdef')  # помилка
```

6. Для обробки об'єктів-множин існують такі методи: *add* вставляє новий елемент в множину, *update* виконує об'єднання, *remove* видаляє елемент за його значенням (викличте функцію *dir*, передавши їй будь□який екземпляр типу *set*, щоб отримати повний перелік всіх доступних методів).

```
my_set = {1, 3, 5}
my_set.update({2, 3, 4})  # my_set |= {2,3,4}
print(my_set)
my_set.intersection_update({0,1,2,3,10})  # my_set &= {0,1,2,3,10}
print(my_set)
my_set.difference_update({1})  # my_set -= {1}
print(my_set)
my_set.symmetric_difference_update({3, 4})  # my_set ^= {3, 4}
print(my_set)
my_set.add(5)  # my_set |= {5}
print(my_set)
my_set.remove(2) #my_set-={2},виводить KeyError, якщо елемента
немає
print(my_set); my_set.discard(2)  # my_set -= {2}
print(my_set)
print(my_set.pop());print(my_set)
my_set.clear();print(my_set)
```

Результат

```
{1, 2, 3, 4, 5}
{1, 2, 3}
{2, 3}
{2, 4}
{2, 4, 5}
{4, 5}
{4, 5}
4
{5}
set()
```

7. Перевірка множин на рівність виконується поелементно, незалежно від типів множин

```
print({1, 2, 3} == frozenset([1, 2, 3]))
print(set('abc') == frozenset('abc'))
print(set('abc') in set([frozenset('abc')]))
```

Результат

```
True
True
True
```

8. Сформувати і вивести множину

1) символів в алфавітному порядку

```
a = set('qwertyuiopasdfghjklzxcvbnm') # формуємо множину
print (a) #виведення множини
b=list(a) # перетворюємо множину у список (її не можна сортувати)
b.sort () #сортуємо список
print (b)  #виведення
```

Результат

```
{'g', 'f', 'u', 'z', 'm', 'n', 'y', 'e', 'd', 'k', 'v', 'l', 's',
'p', 'x', 'h', 'r', 'o', 'i', 'j', 'c', 'w', 'b', 't', 'a', 'q'}

['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

2) цілих чисел у зростаючому порядку з діапазону 1..100, які можна подати у вигляді n^2+m^2, де $n, m \geq 0$.

```
a={k**2+i**2 for i in range(0,8) for k in range(0,8)}
b=list(a); b.sort()
print(b)
```

Результат

```
[0, 1, 2, 4, 5, 8, 9, 10, 13, 16, 17, 18, 20, 25, 26, 29, 32, 34,
36, 37, 40, 41, 45, 49, 50, 52, 53, 58, 61, 65, 72, 74, 85, 98]
```

9. Застосування операцій над множинами, які використовують до списку людей – службовців гіпотетичної компанії:

```
>>> engineers = {‘bob’, ‘sue’, ‘ann’, ‘vic’}
>>> managers = {‘tom’, ‘sue’}
>>> ‘bob’ in engineers # bob – інженер?
True

>>> engineers & managers # хто одночасно є інженером і менеджером?
{‘sue’}
>>> engineers | managers # Всі співробітники з обох категорій
{‘vic’, ‘sue’, ‘tom’, ‘bob’, ‘ann’}
>>> engineers – managers # Інженери, які не є менеджерами
{‘vic’, ‘bob’, ‘ann’}
>>> managers – engineers # Менеджери, які не є інженерами
{‘tom’}
>>> engineers > managers # чи всі менеджериы є інженерами?
False # (надмножина)
>>> {‘bob’, ‘sue’} < engineers
# Обидва співробітника є інженерами? (підмножина)
True
>>> (managers | engineers) > managers
# Множина всіх співробітників є надмножиною менеджерів?
True >>> managers ^ engineers
# Співробітники, які належать до однієї  категорії
{‘vic’, ‘bob’, ‘ann’, ‘tom’}
>>> (managers | engineers) - (managers ^ engineers)
# Перетин!
{‘sue’}
```

Приклад 9.3. Задано два слова. Для кожної літери першого слова (в тому числі для літер, які повторюються в цьому слові) визначити, чи входить літера до складу другого слова. Наприклад, якщо задано слова «*информация*» і «*процессор*», то для літер першого із них відповідь повинна мати вигляд: *нет нет нет да да нет нет да нет нет*.

версія 1

```
import itertools
sl1=input('Введіть слово 1 = ')
sl2=input('Введіть слово 2 = ')
m=len(sl1); n=len(sl2);i=0; L=[]
while m!=i:
  fl=0;
  for j in itertools.count(start=0, step=1):
    if j>=n: break;
    if j<n:
      if sl1[i]==sl2[j]: fl=1
  if fl==1:
        L.append('да ')
  else:
        L.append('нет ')
  i+=1
a=list(L)
pp=''.join(a);print(pp)
```

версія 2

```
sl1=input('Введіть слово 1 = ')
sl2=input('Введіть слово 2 = ')
m=len(sl1)
i=0; L=[]
while m!=i:
  fl=0;
  if sl1[i] in sl2: fl=1
  if fl==1:
        L.append('да ')
  else:
        L.append('нет ')
  i+=1
a=list(L); pp=''.join(a); print(pp)
```

Результат

```
информация
процессор
нет нет нет да да нет нет да нет нет
```

Теоретическая часть

1. МНОЖИНА

Множина – невпорядкований набір унікальних і незмінних об'єктів, який підтримує операції, що відповідають математичній теорії множин. За визначенням, кожний елемент може бути присутнім в множині в одному екземплярі незалежно від того, скільки разів він буде доданий. Оскільки множина є набором інших об'єктів, їй притаманні деякі властивості списків і

словників (множини підтримують ітерації, при необхідності можуть змінюватися в розмірах і містити об'єкти різних типів).

Щоб створити об'єкт множина, потрібно передати послідовність або інший об'єкт, що підтримує можливість ітерацій за його вмістом, вбудованій функції *set*:

```
x=set('abcde'); y=set('bdxyz')
```

Функція повертає об'єкт–множину, який містить всі елементи об'єкта, переданого функції:

```
>>> x
set(['a', 'c', 'b', 'e', 'd'])
```

Множини, створені таким способом, підтримують звичайні математичні операції над множинами за допомогою операторів виразів (перетин, об'єднання, різниця, симетрична різниця множин, перевірка входження в множину, надмножина, підмножина):

```
>>> 'e' in x # Перевірка входження в множину
True
>>> x - y # Різниця множин
set(['a', 'c', 'e'])
>>> x | y # Об'єднання множин
set(['a', 'c', 'b', 'e', 'd', 'y', 'x', 'z'])
>>> x & y # Перетин множин
set(['b', 'd'])
>>> x ^ y # Симетрична різниця (XOR)
set(['a', 'c', 'e', 'y', 'x', 'z'])
>>> x > y, x < y # Надмножина, підмножина
(False, False)
>>> S1 = {1, 2, 3, 4}
>>> S1 & {1, 3} # Перетин
{1, 3}
>>> {1, 5, 3, 6} | S1 # Об'єднання
{1, 2, 3, 4, 5, 6}
>>> S1 - {1, 3, 4} # Різниця
{2}
>>> S1 > {1, 3} # Надмножина
True
```

Для множин визначені такі операції:

+ – об'єднання множин;

– – різниця множин;

* – перетин множин;

= – перевірка еквівалентності двох множин;

<> – перевірка нееквівалентності двох множин;

<= – перевірка, чи є ліва множина підмножиною правої множини;

> = – перевірка, чи є права множина підмножиною лівої множини;

in – перевірка, чи входить елемент, який зображено зліва, в множину, вказану справа.

Результатом операції об'єднання, різниці або перетину є відповідна множина, інші операції дають результат логічного типу. Нехай змінні *x* і *y* зберегли свої значення, присвоєні в попередньому прикладі:

```
>>> x=set('abcde'); y=set('bdxyz')
>>> z=x.intersection(y) # x & y
>>> z
set(['b', 'd'])
>>> z.add('SPAM') # додає один елемент
>>> z
set(['b', 'd', 'SPAM'])
>>> z.update(set(['X', 'Y'])) # об'єднання множин
>>> z
set(['Y', 'X', 'b', 'd', 'SPAM'])
>>> z.remove('b') # видалить один елемент
>>> z
set(['Y', 'X', 'd', 'SPAM'])
```

Будучи ітерованими контейнерами, множини можна передавати функції *len*, використовувати в циклах *for* і в генераторах списків. Однак множини є невпорядкованими колекціями, тому не підтримують операції над послідовностями, такі як індексування і витяг зрізу:

```
>>> for item in set('abc'): print(item * 3)
aaa
ccc
bbb
>>> S = set([1, 2, 3])
>>> S | set([3, 4]) # Оператори виразу вимагають щоб
set([1, 2, 3, 4]) # обидва операнди були множинамии
>>> S | [3, 4]
TypeError: unsupported operand type(s) for |: 'set' and
'list'
>>> S.union([3, 4])
set([1, 2, 3, 4])
>>> S.intersection((1, 3, 5))
set([1, 3])
>>> S.issubset(range(-5, 5))
True
```

Літерали множин. В Python існує можливість використовувати вбудовану функцію *set* для створення множин, при цьому є форма літералів множин, в яких використовують фігурні дужки, раніше зарезервовані для літералів словників. В Python 3.0 такі інструкції є еквівалентними:

```
>>> set([1, 2, 3, 4]) # Виклик функції set
{1, 2, 3, 4} # Літерал множини
>>> set('spam')
{'a', 'p', 's', 'm'}
>>> {1, 2, 3, 4} # Літерали множин
{1, 2, 3, 4}
>>> S = {'s', 'p', 'a', 'm'}
>>> S.add('alot')
```

```
>>> S
{'a', 'p', 's', 'm', 'alot'}
```

Конструкція {} створює пустий словник. Щоб створити пусту множину, треба викликати вбудовану функцію *set*; результат операції виведення пустої множини має інший вигляд:

```
>>> S1 - {1, 2, 3, 4} # Пуста множина
set()
>>> type({}) # Літерал {} позначає пустий словник
<class 'dict'>
>>> S = set() # Ініціалізація пустої множини
>>> S.add(1.23)
>>> S
{1.23}
>>> {1, 2, 3} | {3, 4}
{1, 2, 3, 4}
>>> {1, 2, 3} | [3, 4]
TypeError: unsupported operand type(s) for |: 'set' and 'list'
>>> {1, 2, 3}.union([3, 4])
{1, 2, 3, 4}
>>> {1, 2, 3}.union({3, 4})
{1, 2, 3, 4}
>>> {1, 2, 3}.union(set([3, 4]))
{1, 2, 3, 4}
>>> {1, 2, 3}.intersection((1, 3, 5))
{1, 3}
>>> {1, 2, 3}.issubset(range(-5, 5))
True
```

Множини можуть включати об'єкти тільки незмінних типів: списки і словники не можна додавати в множини, однак можна використати кортежі, якщо з'явиться необхідність зберігати складові значення. В операціях над множинами кортежі порівнюють за своїм повним значенням:

```
>>> S
{1.23}
>>> S.add([1, 2, 3])
# Додаватися можуть тільки незмінні об'єкти
TypeError: unhashable type: 'list'
>>> S.add({'a':1})
TypeError: unhashable type: 'dict'
>>> S.add((1, 2, 3))
>>> S # а ні  список, ні словник не можна додати
{1.23, (1, 2, 3)} # кортеж можна
>>> S | {(4, 5, 6), (1, 2, 3)}
# Об'єднання: теж, що й S.union(...)
{1.23, (4, 5, 6), (1, 2, 3)}
>>> (1, 2, 3) in S
True
>>> (1, 4, 3) in S
False
```

Генератори множин. Конструкцію генератора множин розміщують у фігурні дужки. Генератор множин виконує цикл і збирає результати виразу в кожній ітерації - доступ до значення в поточній ітерації забезпечує змінна циклу. Результатом роботи генератора є нова множина:

```
>>> {x**2 for x in [1, 2, 3, 4]} # Генератор множин
{16, 1, 4, 9}
```

Генератор множин повертає «нову множину, яка містить квадрати значень X, для кожного X зі списку». В генераторах можна також використовувати інші види ітерованих об'єктів, наприклад рядки:

```
>>> {x for x in 'spam'} # Те ж саме, що і: set('spam')
{'a', 'p', 's', 'm'}
>>> {c * 4 for c in 'spam'}
# Множина результатів виразу
{'ssss', 'aaaa', 'pppp', 'mmmm'}
>>> {c * 4 for c in 'spamham'}
{'ssss', 'aaaa', 'hhhh', 'pppp', 'mmmm'}
>>> S = {c * 4 for c in 'spam'}
>>> S | {'mmmm', 'xxxx'}
{'ssss', 'aaaa', 'pppp', 'mmmm', 'xxxx'}
>>> S & {'mmmm', 'xxxx'}
{'mmmm'}
```

Оскільки елементи множин є унікальними, їх можна використовувати для фільтрації повторюваних значень в інших наборах. Для цього досить перетворити набір значень у множину, а потім виконати зворотне перетворення (множина є ітерованим об'єктом, тому її можна передавати функції *list*):

```
>>> L = [1, 2, 1, 3, 2, 4, 5]
>>> set(L)
{1, 2, 3, 4, 5}
>>> L = list(set(L)) # видалення значень, які  повторюються
>>> L
[1, 2, 3, 4, 5]
```