Лекція 14

Множини, діапазони, ітератори

Контрольна робота №4

Правила визначення варіанту

IO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
61	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
62	29	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
63	28		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
64	27	28		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
65	26	27			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Червоний колір – номер варіанту
Зелений колір – група
Чорний колір - номер у списку

Записати результат роботи програми

1	arr = [[5, 2], [7, 4]] for i in arr: i[0] += 3; print(arr)	6	arr = [5, 2, 7, 4] arr = [i * 2 for i in arr] print(arr)

2	arr = [1, 2, 3, 4]
	for i in range(len(arr)-1):
	  arr[i]+=arr[i+1]
	print(arr)
		7	arr = []
			for i in [2, 2, 3, 4]:
			  if i//2==1:arr.append(i+10)
			print(arr)

3	arr = [1, 5, 12, 45] j=(i for i in arr if i % 5 == 0) print(sum(j))	8	arr = [[10, 20], [3, 4]] arr1=[j * 10 for i in arr for j in i] print(arr1)

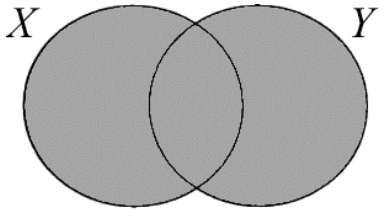
4	def func(elem):
	  return elem -3
	arr = [1, 1, 1, 2, 2, 2, 1, 2, 3, 4, 4]
	print(set(map(func, arr)),end=" ")
		9	def func(elem):
			  return elem * 2
			arr = [2, 4, 6, 8]; j=map(func, arr)
			for i in arr: print(next(j), end=" ")

5	arr=[5, 2, 7, 4] for i in arr: i += 50 print(arr)	10	def func(elem): return elem + 10 arr = [0, 2, 3, 4, 5] print(tuple(map(func, arr)),end=" ")

11 27	def func(e1,e2,e3): return e1-e2+e3 arr1 = [10, 20, 30, 40, 50] arr2 = [5, 10]; arr3 = [1, 2, 3, 4, 5] print(list(map(func, arr1, arr2, arr3)))	19 28	a = [1, 2, 3, 4, 5]; b = [10, 20, 30] c = [100, 200, 300, 400] d=[max(x,y,z) for (x,y,z) in zip(a,b,c)] print(d)
12	J=zip([1, 2, 3], [4, 5, 6], [7, 8, 9]) print(max(list(J)[0]))	20 29	>>> J=zip([1, 2, 3], [4, 5, 6], [7, 8, 9]) >>> next(J)
13	def func(elem): return elem >= 0 a = [-1, 2, -3, 4, 0, -20, 10] a = [i for i in a if func(i)]; print(a)	21	def func(elem): return elem < 0 arr = [-1, 2, -3, 4, 0, -20, 10] arr = list(filter(func, arr)); print(arr)
14	>>> list(zip([1,2,3,5], [4, 6], [7, 8, 9]))	22	>>> list(filter(None, [1, 0, None, [], 2]))
15	>>> arr = [1, 2, 3] >>> arr.extend([4, 5, 6]); arr	23	>>> arr = [1, 2, 3] >>> arr. append([5, 6]); arr
16	>>> arr = [1, 2, 3] >>> arr[len(arr):] = [4, 5, 6]; arr	24	a = [1, 2, 3];b = [4, 5, 6] j = list(zip(a,b)); print(j, sum(j[2]))
17	>>> arr = [1, 2, 3] >>> arr.insert(2, 100); arr	25	>>> arr = [1, 2, 3] >>> arr[:0] = [-2, -1, 0]; arr
18	>>> arr = [1, 2, 3, 4, 5]; arr.pop() >>> arr	26	>>> arr = [1, 2, 3, 4, 5] >>> del arr[:2]; arr

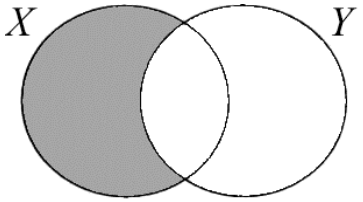
Оператори **`|, union(),`**
`a |= b, a.update(b)`

a={1,2,3}; b={1,2,4}; c=a | b; print(c); a |=b; print(a)
{1,2,3,4}



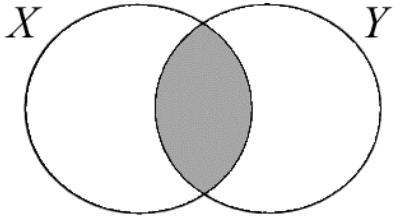
Оператори **`-, difference(),`**
`a -= b` і **`a.difference_update(b)`**

a={1,2,3}; b={2,4}; c=a - b; print(c); a -=b; print(a)
{1,3}



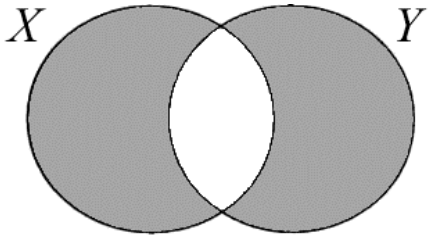
Оператори **`&, intersection()`**
`a &= b` і **`a.intersection_update b`**

a={1,2,3}; b={1,2,4}; c=a & b; print(c); a &=b; print(a)
{1,2}



Оператори для роботи з множинами (продовження)

Оператор **^** і **symmetric_difference()** – повертають усі елементи обох множин, крім елементів, які містяться в обох цих множинах:



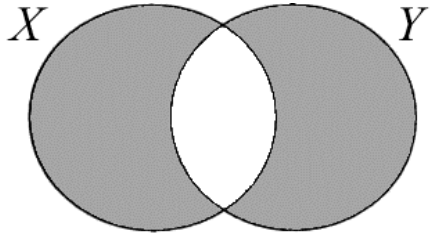
Приклад 1.

```
>>> s = {1, 2, 3}
>>> s ^ {1, 2, 4} #{1, 2, 3}^{1, 2, 4}
{3, 4}
>>> s.symmetric_difference(set([1, 2, 4]))
{3, 4}

s1= set ([1, 2, 3])
s2= set ([1, 2, 4])
s3=s1^s2
```

Продовження прикладу 1

```
>>> s = {1, 2, 3}
>>> s ^ {1, 2, 3} #{1, 2, 3}^{1, 2, 3}
set()

>>> s.symmetric_difference(set([1, 2, 3]))
set()

>>> s ^ set([4, 5, 6]) #{1, 2, 3}^{4, 5, 6}
{1, 2, 3, 4, 5, 6}

>>> s.symmetric_difference(set([4, 5, 6]))
{1, 2, 3, 4, 5, 6}

s = set(range(4)) ^ set(range(2,7))
print(s)
{0, 1, 4, 5, 6}
```

Оператори **a ^= b** і a.symmetric_difference_update(b)
У множині **a** будуть усі елементи обох множин, крім тих, що містяться в обох цих множинах:



Приклад 2.

```
>>> s ={1, 2, 3}
>>> s.symmetric_difference_update(set([1, 2, 4]))
>>> s
{3, 4}
# Зараз s містить множину {3,4}
>>> s ^= set([3, 5, 6])
>>> s
{4, 5, 6}
```

Оператори порівняння множин:

Оператор **in** – перевірка наявності елемента в множині:

Приклад 3.

```
>>> s = set([1, 2, 3, 4, 5])
>>> 1 in s, 12 in s
(True, False)
```

Оператор **not in** – перевірка відсутності елемента в множині:

Приклад 4.

```
>>> s = {1, 2, 3, 4, 5}
>>> 1 not in s, 12 not in s
(False, True)
```

Оператор == – перевірка на рівність:

Приклад 5.

```
>>> set ( [1, 2, 3]) == set ( [1, 2, 3])
True
>>> set ( [1, 2, 3]) == set ( [3, 2, 1] )
True
>>> set ( [1, 2, 3]) == set ( [ 1, 2, 3, 4] )
False
>>> set(["a", "b", "c"]) == set(["a", "b", "c"])
True
>>> set(["a", "b", "c" ])=={"a", "b", "c", "d"}
False
>>> set(range(4))=={0,1,2,3}
True
res=set(range(4))==set(range(2,7))
print(res)
Результат роботи: False
```

Оператори `a <= b` і `a.issubset(b)` – перевіряють, чи входять усі елементи множини `a` в множину `b`.
Множина `a` може дорівнювати множині `b`.

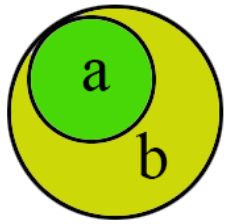
Приклад 6.

```
>>> s = set ([1, 2, 3])
>>> s <= set([1, 2, 3])
True
>>> s <= set([1, 2]),
False
>>> s <= set([1, 2, 3, 4])
True

>>> s = set ([1, 2, 3, 4])
>>> s.issubset(set([1, 2]))
False
>>> s.issubset(set([1, 2, 3, 4, 5]))
True
```

Оператор **a < b** – перевіряє, чи строго входять усі елементи множини a в множину b, причому множина a не повинна дорівнювати множині b:

Приклад 7.

```
>>> s = set([ 1, 2, 3])
>>> s < set([1, 2, 3])
False
>>> s < set([1, 2, 3, 4])
True
>>> s = set (["a", "b", "c"])
>>> s < set(["a", "b", "c"])
False
>>> s < set(["a", "b", "c", "d"])
True

>>> set(range(3)) < set(range(4))
True
```



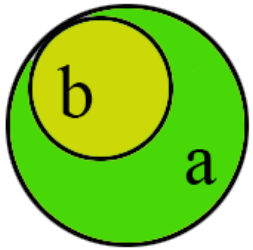
Оператори **a >= b** і **a.issuperset(b)** – перевіряють, чи не строго входять усі елементи множини b у множину a:

Приклад 8.

```
>>> s = set([1, 2, 3])
>>> s >= set([1, 2]), s >= set([1, 2, 3, 4])
(True, False)

>>> s.issuperset(set([1,2]))
True
>>> s.issuperset(set([1, 2, 3, 4]))
False

>>> s = set (["a", "b", "c"])
>>> s.issuperset(set(["a", "b"]))
True
s >= set(["a", "b"])
True
```

Оператор **a > b** – перевіряє, чи входять усі елементи множини b у множину a, причому множина a не повинна дорівнювати множині b:

Приклад 9.

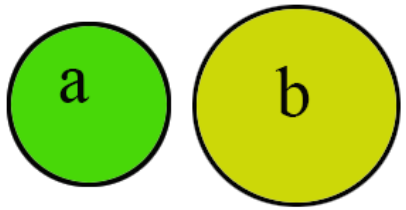



```
>>> s = set([1, 2, 3])
>>> s > set([1, 2])
True
>>> s > set([1, 2, 3])
False
>>> s > set([1, 2, 3, 4])
False

>>> s = set (["a", "b", "c"])
>>> s > set(["a", "b", "c"])
False
>>> s > set(["a", "b"])
True
```

Оператор **a.isdisjoint(b)** – перевіряє, чи є множини a й b повністю різними, тобто не утримуючими жодного співпадаючого елемента:

Приклад 10.

```
>>> s = set([1, 2, 3])
>>> s.isdisjoint(set([4, 5, 6]))
True
>>> s.isdisjoint(set([1, 3, 5]))
False

>>> s = set (["a", "b", "c"])

>>> s.isdisjoint(set(["a", "b"]))
False

>>> s.isdisjoint(set(["d", "e"]))
True
```



Методи для роботи з множинами

Метод **copy()** – створює копію множини.
Примітка!!!!: оператор **=** присвоює лише посилання на той же об'єкт, а не копіює його.

Приклад 11.

```
>>> s = set([1, 2, 3])
>>> c=s; s is c # За допомогою = копію створити не можна!
True

>>> c = s.copy() # Створюємо копію об'єкта
>>> c
{1, 2, 3}
>>> s is c       # Тепер це різні об'єкти
False
```

Метод **add (<Елемент>)** – додає <Елемент> у множину:

Приклад 12.

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

my = {"Petrenko"}
my.add("Ivan")
print (my)
my.add("Ivanovich")
print (my)

Результат:
{'Ivan', 'Petrenko'}
{'Ivanovich', 'Ivan', 'Petrenko'}
```

```
my= {"Petrenko"}
my^= {"Ivan"}
my^= {"Ivanovich"}
print (my)
```

Метод **remove(<Елемент>)** – видаляє <Елемент> із множини. Якщо елемент не знайдений, то виконується виключення Keyerror:

Приклад 13.

```
>>> s = set ([1, 2, 3] )
>>> s.remove(3); s # Елемент існує
{1, 2}
>>> s.remove(5) # Елемент НЕ існує
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Keyerror: 5

my = {"Petrenko", "Sydorenko"}
my.remove("Sydorenko")
print(my)
Результат: {'Petrenko'}
```

Метод **discard (<Елемент>)** – видаляє <Елемент> із множини, якщо він присутній. Якщо зазначений елемент не існує, ніякого виключення не виконується:

Приклад 14.

```
>>> s = set([1, 2, 3])
>>> s.discard(3); s # Елемент існує
{1, 2}
>>> s.discard(5); s # Елемент НЕ існує
{1, 2}

my = {"Petrenko", "Sydorenko"}
print(my)
my. discard ("Ivanov")
print(my)
Результат:
{'Petrenko', 'Sydorenko'}
{'Petrenko', 'Sydorenko'}
```

Метод **pop ()** – видаляє елемент із множини й повертає його. Якщо елементів немає, то виконується виключення Keyerror. Увага! Тип «set» не підтримує індексацію елементів!!!! Тому метод pop використовується без параметрів.

Приклад 15.

```
>>> s = set([1, 2])
>>> s.pop()
1
>>> s
{2}
>>> s.pop()
2
>>> s
set()
>>> s.pop() # Якщо немає елементів, то помилка
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Keyerror: 'pop from an empty set'
```

Метод **`clear()`** – видаляє всі елементи із множини:

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

```
my = {"Petrenko", "Sydorenko"}
my.clear()
print("Множина:",my, "Довжина множини:",len(my))
```

```
Результат:
Множина: set() Довжина множини: 0
```

Генератори множин

1. Синтаксис генераторів множин схожий на синтаксис генераторів списків
2. Відмінність у тому, що вираз міститься у фігурних дужках, а не у квадратних.

Розглянемо приклад, де результатом є множина, у якій всі повторювані елементи будуть вилучені.

Приклад 16.

```
>>> {x for x in [1, 2, 1, 2, 1, 2, 3]}
{1, 2, 3}

>>> {x for x in ["a", "b", "b", "a", "c"]}
{'b', 'c', 'a'}

>>> {x for x in ["winter", "summer", "fall",
"fall", "spring"]}
{'summer', 'fall', 'spring', 'winter'}
```

Генератори множин зі складною структурою

1. Генератори множин можуть складатися з декількох вкладених циклів **for**
2. Можуть містити оператор розгалуження **if** після циклу.

Приклад 17. Унікальні парні елементи

```
>>> {x for x in [1,2,1,2,1,2,3] if x % 2 == 0}
{2}

>>> {x for x in [1,2,1,2,1,2,3] if x < 3}
{1, 2}

>>> {x for x in [1,2,1,2,1,2,3] if (x<3)&(x>1)}
{2}

>>> {x for x in list(zip([1,2],[1,2],[1,2,3])) if x[0] % 2 == 0}
{(2, 2, 2)}
```

Тип множин **frozenset**. На відміну від типу `set`, множину типу `frozenset` не можна змінити. Оголосити множину можна за допомогою функції `frozenset()`:

Приклад 18

```
>>> f = frozenset()
>>> f
frozenset()
```

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

Приклад 19

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

```
>>> frozenset([1, 2, 3, 4, 4]) # Перетворимо список
frozenset({1, 2, 3, 4})

>>> >>> frozenset((1, 2, 3, 4, 4))# Перетворимо кортеж
frozenset({1, 2, 3, 4})
```

Множини `frozenset` підтримують оператори, які не змінюють саму множину, а також наступні методи:

```
copy(),
difference(),
intersection(),
issubset(),
issuperset(),
symmetric_difference()
union().
```

Діапазони

1. Діапазони – послідовності цілих чисел з заданими початковим і кінцевим значенням і кроком (проміжком між сусідніми числами).
2. Мають властивості, подібні до списків, кортежів і множин.
3. Діапазони – незмінювані послідовності, подібно до кортежів.

4. Найважливіша перевага діапазонів – компактність.
Незалежно від кількості у діапазоні елементів-чисел, діапазон завжди займає той самий обсяг оперативної пам'яті.

5. Недолік. У діапазон можуть входити лише числа, які послідовно ідуть одне за одним.

6. Область використання: для перевірки входження значення в будь-який інтервал і для організації циклів.

Створення діапазонів

Для створення діапазону застосовується функція `range()`:

```
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`.

Приклад 20

```
>>> r = range(1, 10)
>>> for i in r: print(i)
1 2 3 4 5 6 7 8 9

>>> r = range(10, 110, 10)
>>> for i in r: print(i)
10 20 30 40 50 60 70 80 90 100

>>> r = range(10, 1, -1)
>>> for i in r: print(i)
10 9 8 7 6 5 4 3 2
```

Перетворити діапазон у список, кортеж, звичайну або незмінювану множину можна за допомогою функцій `list(),tuple(), set()` або `frozenset()` відповідно:

Приклад 21

```
>>> list(range(1, 10)) # Перетворимо в список
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> tuple(range(1, 10)) # Перетворимо в кортеж
(1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> set(range(1, 10)) # Перетворимо в множину
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Діапазони підтримують:
- доступ до елемента по індексу,
- одержання зрізу (у результаті повертається також діапазон),
- перевірку на входження
- перевірку на невходження,
- функції `len()`, `min()`, `max()`,
- методи `index()` і `count()`.

Приклад 22

```
>>> r = range(1, 10)
>>> r[2], r[-1]
(3, 9)

>>> r[2:4]
range(3, 5)

>>> 2 in r, 12 in r
(True, False)

>>> 3 not in r, 13 not in r
(False, True)

>>> len(r), min(r), max(r)
(9, 1, 9)
>>> r.index(4), r.count(4)
(3, 1)
```

Оператори порівняння діапазонів

Оператор **==** – повертає `True`, якщо діапазони рівні, і `False` якщо ні.

Діапазони вважаються рівними, якщо вони містять однакові послідовності чисел!!

Приклад 23

```
>>> range(1, 10) == range(1, 10, 1)
True

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

```
>>> range(1, 10, 2) == range(1, 11, 2)
True
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
>>> list(range(1, 11, 2))
[1, 3, 5, 7, 9]


>>> range(1, 10, 2) == range(1, 12, 2)
False
>>> list(range(1, 10,2))
[1, 3, 5, 7, 9]
>>> list(range(1, 12,2))
[1, 3, 5, 7, 9, 11]
```

Оператор `!=` – повертає `True`, якщо діапазони не рівні, і `False` в протилежному випадку:

Приклад 24

```
>>> range(1, 10, 2) != range(1, 12, 2)
True
>>> list(range(1, 10,2))
[1, 3, 5, 7, 9]
>>> list(range(1, 12,2))
[1, 3, 5, 7, 9, 11]

>>> range(1, 10) != range(1, 10, 1)
False
>>> list(range(1, 10))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 10,1))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Властивості `start`, `stop` і `step`, що повертають, відповідно

`start` – початкову границю діапазону,

`stop` – кінцеву границю діапазону,

`step` – крок діапазону.

Приклад 25

```
>>> r = range(1, 10)
>>> r.start, r.stop, r.step
(1, 10, 1)

>>> r = range(1, 11, 2)
>>> r.start, r.stop, r.step
(1, 11, 2)
```

Модуль `itertools`

Модуль **`itertools`** містить функції, що дозволяють:
- генерувати різні послідовності на основі інших послідовностей,
- виконувати фільтрацію елементів і ін.

Усі функції повертають об'єкти, що підтримують ітерації (`ітератори`).

Перш ніж використовувати функції, необхідно підключити модуль за допомогою інструкції:

```
import itertools
```

Генерація невизначеної кількості значень

Для генерації невизначеної кількості значень призначені наступні функції:

Функція **`count([start=0][, step=1])`**.
Створює нескінченно наростаючу послідовність значень. Початкове значення задають параметром `start`, а крок – параметром `step`.

Приклад 26

```
import itertools
for i in itertools.count():
    if i > 10: break
    print(i, end=" ")
Результат: 0 1 2 3 4 5 6 7 8 9 10
```

```
import itertools
p = list(zip(itertools.count(), "абвгд"))
print(p)
```

Результат:

```
[(0,'а'), (1,'б'), (2,'в'), (3,'г'), (4,'д')]
```

```
import itertools
p = list(zip(itertools.count(start=2, step=2),
"абвгд"))
print(p)
```

Результат:

```
[(2,'а'), (4,'б'), (6,'в'), (8,'г'), (10,'д')]
```

Функція **cycle(<Послідовність>)**

На кожній ітерації повертається черговий елемент послідовності.

Коли буде досягнутий кінець послідовності, перебір почнеться спочатку, і так нескінченно.

Приклад 27

```
import itertools
n = 1
for i in itertools.cycle("абв"):
    if n > 10: break
    print(i, end=" ")
    n += 1
Результат:
а б в а б в а б в а
```

```
import itertools
p = list(zip(itertools.cycle([0, 1]), "абвгд"))
print(p)
```

Результат:

[(0, 'а'), (1, 'б'), (0, 'в'), (1, 'г'), (0, 'д')]

```
import itertools
p = list(zip(itertools.cycle(["a", "b", "v",
"g", "d"]), "абвгд"))
print(p)
```

Результат:

[('a', 'а'), ('b', 'б'), ('v', 'в'), ('g', 'г'), ('d', 'д')]

Функція **repeat(<Об'єкт>[, <Кількість повторів>])**
Повертає об'єкт зазначену кількість раз. Якщо кількість повторів не зазначена, то об'єкт повертається нескінченно.

Приклад 28

```
import itertools
p = list(itertools.repeat(1,10))
print(p)
Результат:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

p = list(zip(itertools.repeat(5), "абвгд"))
print(p)
Результат:
[(5, 'а'), (5, 'б'), (5, 'в'), (5, 'г'), (5, 'д')]
```

Ітератори, що підтримують скінченні послідовності

https://docs.python.org/3/library/itertools.html

```
from itertools import*
Функція chain(p,q,l,…)
p=[1,2,3,]
q=["a","b","c","d"]
l=[7,8]
print(list(chain(p,q,l)))
Результат: [1, 2, 3, 'a', 'b', 'c', 'd', 7, 8]
```

```
Функція compress(data,selector)
data="ABCDE"
selector=[1,0,True,0,23]
print(list(compress(data,selector)))
Результат: ['A', 'C', 'E']
```

Функція islice(seq, [start,] stop [, step]**)**

```
sec="ABCDE"
print(list(islice(sec,0,4,2))
```

product(seq, repeat=2) (Комбінаторні функції)

```
print(list(product('AB',
repeat=2)))[('A','A'), ('A','B'), ('B','A'),
('B','B')]
```