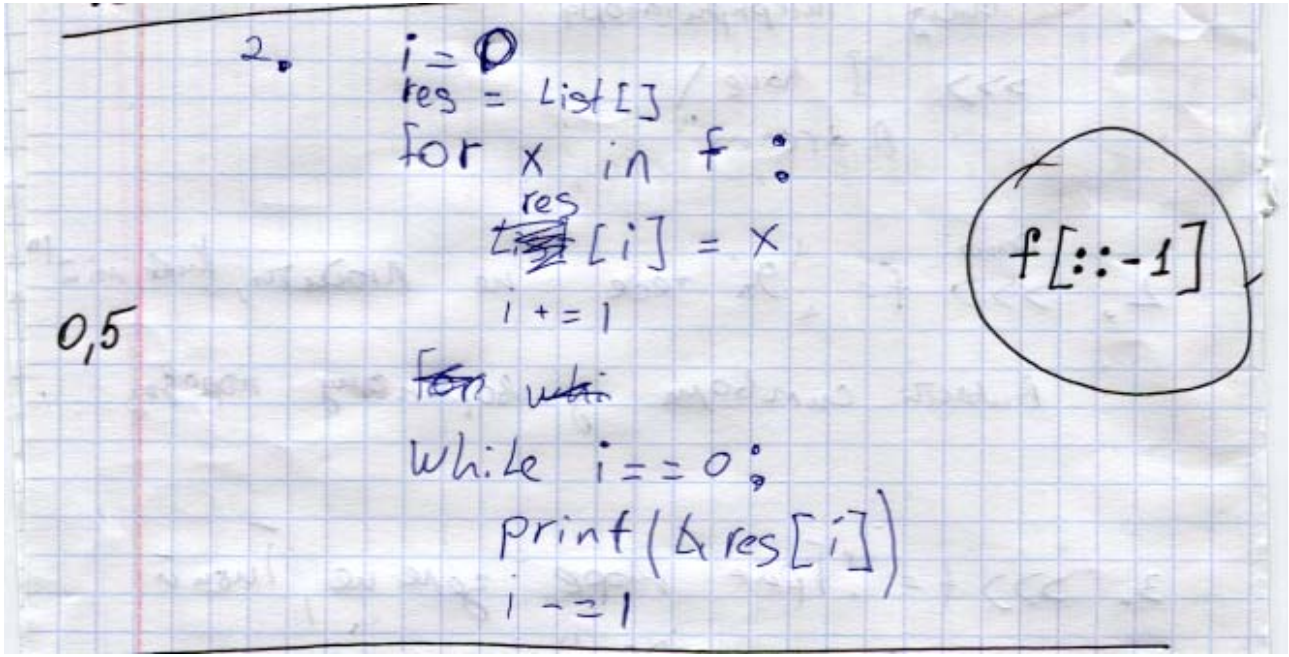
Лекція 12
Операції зі списками в Python

Написати інструкцію для
будь-якого рядка!
А даний рядок – як контрольний О
3. Дано рядок приклад!
⇝ в = "Грає море зелене, Тихий день догора"
1) Відповідь:
річч ('ГРАЄ МОРЕ ЗЕЛЕНЕ, ТИХИЙ ДЕНЬ
ДОГОРА')



f="рядок" res=list(f) res.reverse() print("" .join(res))	f="рядок" print("".join(reversed(f)))	f="рядок" print(f[::-1])

Перебір елементів списку

Перебрати всі елементи списку можна за допомогою циклу for:

Приклад 1.

```
arr = [1, 2, 3, 4, 5, 6]
for i in arr: print(i, end=" ")
Результат:
1 2 3 4 5 6

arr = ["U","k", "r", "a", "i", "n", "e"]
for i in arr: print(i, end=" ")
Результат:
U k r a i n e
```

Зміна змінної циклу

1. Змінну і усередині циклу можна змінити.
2. Якщо змінна циклу посилається на незмінюваний тип даних (наприклад, число або рядок), то це не позначиться на початковому списку:

Приклад 2.

```
# Елементи мають незмінюваний тип (число)
>>>arr=[1, 2, 3, 4, 5]
>>> for i in arr: i += 10
>>> arr
[1, 2, 3, 4] # Список не змінився

# Елементи мають змінюваний тип (список)
>>> arr = [[1, 2], [3, 4]]
>>> for i in arr: i[0] += 10
>>> arr # Список змінився
[ [11, 2], [13, 4]]
```

Функція range()

Функція `range ()` має наступний формат:

range([<Початок>,]<Кінець>[,<Крок>])

1. Перший параметр задає початкове значення. Якщо параметр <Початок> не зазначений, то за замовчуванням використовується значення 0.
2. У другому параметрі <Кінець> вказується кінцеве значення. Це значення не входить у діапазон значень, що повертається.
3. Якщо параметр <Крок> не зазначений, то використовується значення 1.

Функція `range()` використовується для одержання доступу до кожного елемента списку. Функція повертає об'єкт-діапазон, що підтримує ітерації, а за допомогою діапазону усередині циклу `for` можна одержати поточний індекс.

Приклад застосування функції `range()`

Приклад 3.

Помножимо кожний елемент списку на 2:

```
arr = [1, 2, 3, 4]
for i in range(len(arr)):
    arr[i] *= 2
print (arr)
# Результат виконання: [2, 4, 6, ]
```

```
arr = [1, 2, 3, 4, 5, 6, 7, 8]
for i in range(len(arr)):
    if i>0: arr[i]+=arr[i-1]
print (arr)
# Результат виконання:
[1, 3, 6, 10, 15, 21, 28, 36]
```

i	Дія	arr
0	-	1
1	2+1	3
2	3+3	6
3	4+6	10
4	5+10	15
5	6+15	21
6	7+21	28
7	8+28	36

for i **in** range(1,len(arr)):

Функція enumerate

Можна також скористатися функцією

```
enumerate(<Об'єкт>[,start=0]),
```

Повертає кортеж з індексу й значення поточного елемента списку

<Об'єкт> – повинен підтримувати ітерації

[start] – початкове значення ітератора

Приклад 4.

```
>>> seasons=['Spring','Summer','Fall','Winter']
>>> list(enumerate(seasons))
[(0,'Spring'),(1,'Summer'),(2,'Fall'),(3,'Winter')]

>>> list(enumerate(seasons, start=1))
[(1,'Spring'),(2,'Summer'),(3,'Fall'),(4, 'Winter')]
```

Приклад застосування функції `enumerate()`

Помножимо кожний елемент списку на 2:

Приклад 5.

```
arr = [1, 2, 3, 4]
for i, elem in enumerate(arr):
    elem *=2 # елемент arr не змінюється
    print(elem, end = " ")
    arr[i] *= 2 # елемент arr змінюється
print (arr)
```

```
Результат обчислень:
2 4 6 8 [2, 4, 6, 8]
```

Використання циклу `while`

1. Перебрати елементи можна за допомогою циклу `while`.
2. Слід пам'ятати, що цикл `while` працює повільніше від циклу `for`.

Помножимо кожний елемент списку на 2, використовуючи цикл `while`:

Приклад 6.

```
arr = [1, 2, 3, 4]
i, c = 0, len(arr)
while i < c:
    arr[i] *= 2
    i += 1
print(arr) # Результат виконання: [2, 4, 6, 8]
```

Генератори списків

У прикладі 3 ми змінювали елементи списку таким чином:

```
arr = [1, 2, 3, 4]
for i in range(len(arr)):
      arr[i) *= 2
print(arr) # Результат виконання: [2, 4, 6, 8]
```

Такий перебір можна зробити швидше, використавши генератор

Переваги використання генераторів:

1. За допомогою генераторів списків той же самий код можна записати більш компактно.
2. Генератори списків працюють швидше за цикл `for`.

Відмінність

1.Замість зміни початкого списку повертається новий список.

Генератор списку для перебору має формат:

```
<Новий список>=[<Інструкція> for ітератор in
<Початковий список>]
```

1. Інструкція, виконувана усередині циклу, міститься перед циклом.
2. Вираз усередині циклу не містить оператора присвоювання
3. На кожній ітерації циклу буде генеруватися новий елемент, якому неявним чином присвоюється результат виконання виразу усередині циклу.
4. У підсумку буде створений новий список, що містить змінені значення елементів початкового списку.

Приклад 7. Використання генератора

```
arr = [ 1, 2, 3, 4]
arr = [ i * 2 for i in arr]
print(arr) # Результат Виконання: [2, 4, 6, 8]
```

Генератори списків зі складною структурою

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

Одержимо парні елементи списку й помножимо їх на 10:

Приклад 8.

```
arr = [1, 2, 3, 4]
arr = [ i * 10 for i in arr if i % 2 == 0]
print(arr) # Результат виконання: [20, 40]
```

Приклад 9. Еквівалентний код:

```
arr = []
for i in [1, 2, 3, 4] :
    if i % 2 == 0:  # Якщо число парне
        arr .append(i*10) # Додаємо елемент
print(arr) # Результат виконання: [20, 40]
```

Ускладнимо приклад

Одержимо парні елементи вкладеного списку й помножимо їх на 10:

Приклад 10.

```
arr = [[1, 2], [3, 4], [5, 6]]
arr1=[j * 10 for i in arr for j in i if j % 2 == 0]
print(arr1) # Результат виконання: [20, 40, 60]
```

Приклад 11. Еквівалентний код:

```
arr = [[1, 2], [3, 4], [5, 6]]
arr1 = []
for i in arr:
    for j in i:
        if j % 2 == 0: # Якщо число парне
            arr1.append(j * 10) # Додаємо елемент
print (arr)
# Результат виконання: [20, 40, 60]
```

Вирази-генератори

1. Якщо вираз розмістити усередині не в квадратних, а в круглих дужках, то буде повертатися не список, а ітератор.
2. Такі конструкції називають виразами-генераторами.

Приклад 12.

```
>>> arr = [1, 4, 12, 45, 10]
>>>j=(i for i in arr if i % 2 == 0)
>> next(j)
4
>>> next(j)
12
>>> next(j)
10
>>> next(j)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
StopIteration
```

Застосування виразу генератора у функції sum

Приклад 13.

```
>>> arr = [1, 4, 12, 45, 10]
>>> j=(i for i in arr if i % 2 == 0)
>>> sum(j)
26
```

У прикладі використана функція

```
sum(<елемент послідовності>[, start])
```

Функція sum додає елементи списку зліва направо, починаючи з нульового елемента (за замовчуванням).

```
>>> arr = [1, 4, 12, 45, 10]
>>> sum(arr)
72
```

Функція `map()`

Вбудована функція `map()` дозволяє застосувати функцію до кожного елемента послідовності. Функція має наступний формат:

```
map(<Функція>, <Послідовність1>[, ... ,
                <ПослідовністьN>))
```

1. Функція **`map()`** повертає ітератор (індекс)

3. Параметр `<Функція>` містить посилання на функцію, у яку буде передаватися поточний елемент послідовності.

Приклад використання функції `map()`

Додамо до кожного елемента `elem` списку число 10.

Приклад 14. Використаємо конструкцію: ітератор+next

```
def func(elem):
    """ Збільшення значення кожного елемента списку """
    return elem + 10 # Повертаємо нове значення
arr = [1, 2, 3, 4, 5]
j=map(func, arr)
for i in arr: print(next(j, end=" ")
результат: 11 12 13 14 15
```

Приклад 15. Використаємо функцію `list()`

```
def func(elem):
    """ Збільшення значення кожного елемента списку """
    return elem + 10 # Повертаємо нове значення
arr = [1, 2, 3, 4, 5]
print(list(map(func, arr)), end="")
```

Приклад 16. Використаємо функцію `tuple()`

```
def func(elem):
    return elem * 10
arr = [1, 2, 3, 4, 5]
print(tuple(map(func, arr)), end=" ")
```

Приклад 17. Використаємо функцію `set()`

```
def func(elem):
    return elem - 10
arr = [1, 2,2, 2, 2, 2, 2, 2, 3, 4, 5]
print(set(map(func, arr)), end=" ")
```

Приклад 18. Використаємо функцію `sum()`

```
def func(elem):
    return elem**2
arr = [1, 2, 3, 4, 5]
J= map(func, arr)
print(sum(J))
```

`map` з декількома послідовностями

Функції `map()` можна передати кілька послідовностей.

Для цього випадку функція повинна повертати стільки ж елементів, скільки маємо послідовностей у параметрах функції map()

У функцію зворотного виклику будуть передаватися одночасно елементи, які розташовані у послідовностях на однаковому зсуві.

Знайдемо суму елементів трьох списків

Приклад 19.

```
def func(e1, e2, e3):
    return e1 + e2 + e3 # Повертаємо нове значення
 arr1 = [1, 2, 3, 4, 5]
 arr2 = [10, 20, 30, 40, 50]
 arr3 = [100, 200, 300, 400, 500]
 print(list(map(func, arr1, arr2, arr3)))
 # Результат виконання: [111, 222, 333, 444, 555]
```

Приклад 20.

```
def func(e1, e2, e3):
    return e1 - e2 - e3
arr1 = [100, 200, 300, 400, 500]
arr2 = [10, 20, 30, 40, 50]
arr3 = [1, 2, 3, 4, 5]
print(list(map(func, arr1, arr2, arr3)))
# Результат виконання: [89, 178, 267, 356, 445]
```

`map` і кілька послідовностей різної довжини

Якщо кількість елементів у послідовностях буде різним, то в якості обмеження вибирається послідовність із мінімальною кількістю елементів:

Приклад 21.

```
def func(e1, e2, e3):
    """ Додавання елементів трьох різних списків """
    return e1 + e2 + e3
arr1 = [1, 2, 3, 4, 5]
arr2 = [10, 20]
arr3 = [100, 200, 300, 400, 500]
print(list( map(func, arr1, arr2, arr3)))
# Результат виконання: [111, 222]
```

Вбудована функція `zip()`

Формат функції:

`zip`(<Послідовність1>[, ... ,<ПослідованістьN>])

1. На кожній ітерації повертає кортеж, що містить елементи послідовностей, які розташовані на однаковому зсуві.

Приклад 22.

```
>>> J=zip([1, 2, 3], [4, 5, 6], [7, 8, 9])
>>> next(J)
(1, 4, 7)
>>> next(J)
(2, 5, 8)
>>> next(J)
(3, 6, 9)
>>> next(J)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
StopIteration
```

Використання zip з функцію list().

Приклад 22.

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

Приклад 23.

```
a = [1, 2, 3]
b = [4, 5, 6]
c = [7, 8, 9]
j = list(zip(a,b,c))
print("Список j =",j)
s1,s2,s3 = sum(j[0]),sum(j[1]),sum(j[2])
print("Суми елементів з однаковим зсувом
=",s1,s2,s3)
Результат:
Список j = [(1, 4, 7), (2, 5, 8), (3, 6, 9)]
Суми елементів з однаковим зсувом = 12 15 18
```

Приклад 24.

```
J=zip([1, 2, 3], [4, 5, 6], [7, 8, 9])
for i in range(3):
    a=next(J)
    sum(a)
    print(a, sum(a))
```

```
Результат:
(1, 4, 7) 12
(2, 5, 8) 15
(3, 6, 9) 18
```

Приклад 25.

```
J=zip([1, 2, 3], [4, 5, 6], [7, 8, 9])
print(max(list(J)[0]))
```

Результат: 7

`zip` і кілька послідовностей різної довжини

Якщо кількість елементів у послідовностях буде різною, то в результат потраплять тільки елементи, які існують у всіх послідовностях на однаковому зсуві:

Приклад 26.

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

Приклад 27

```
a=[1,2,3,4]
b=[5,6]
c=[9,10,11]
J = zip(a,b,c)
z=min(len(a),len(b),len(c))
for i in range(z):
    y=next(J)
    print(y)
```

Змінимо програму додавання елементів трьох списків з використанням функції `zip()` замість функції `map()`.

Приклад 28.

```
arr1 = [1, 2, 3, 4, 5]
arr2 = [10, 20, 30, 40, 50]
arr3 = [100, 200, 300, 400, 500]
arr=[x + y + z for (x, y, z) in zip(arr1, arr2, arr3)]
print(arr)
# Результат виконання: [111, 222, 333, 444, 555]
```

Приклад 29.

```
arr1 = [1, 2, 3, 4, 5]
arr2 = [10, 20, 30, 40, 50]
arr3 = [100, 200, 300, 400]
arr=[min(x,y,z) for (x, y, z) in zip(arr1,arr2,arr3)]
print(arr)
Результат:[1, 2, 3, 4]
```

Функція `filter()`

Функція `filter()` дозволяє виконати перевірку елементів послідовності.

Формат функції:

```
filter(<Функція>, <Послідовність>)
```

1. Якщо в першому параметрі замість назви функції вказати значення `None`, то кожний елемент послідовності буде перевірений на відповідність значенню `True`.

2. Якщо елемент у логічному контексті повертає значення `False`, то він не буде доданий в результат, що повертається.

3. Функція повертає ітератор – об'єкт, що підтримує ітерації.

4. Щоб одержати список, необхідно результат передати у функцію `list()`.

Приклад 30. Використання функції `filter`

```
>>> J = filter(None, (1, 0, None, [], 2))
>>> next(J)
1
>>> next(J)
2
>>> next(J)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
StopIteration

>>> list(filter(None, [1, 0, None, [], 2]))
[ 1, 2]
```

Аналогічна операція з використанням генераторів списків має такий вигляд:

Приклад 31.

```
>>> [i for i in [1, 0, None, [], 2] if i]
[ 1, 2]
```

Filter + функція

У прикладі 30 замість параметра **`None`** можна вказати посилання на функцію.
У цю функцію як параметр буде передаватися поточний елемент послідовності.

Якщо елемент потрібно додати в значення,

що повертається функцією `filter()`,

то усередині функції зворотного виклику слід повернути значення `True`,

а якщо ні, то – значення `False`.

Приклад 32. Вилучимо всі від'ємні значення зі списку.

Варіант використанням `next`

```
def func(elem):
    return elem >= 0
arr = [-1, 2, -3, 4, 0, -20, 10]
J=filter(func,arr)
while True:
    try:
        a=next(J)
        print(a)
    except: StopIteration
```

Варіант використанням `list`

```
def func(elem):
    return elem >= 0
arr = [-1, 2, -3, 4, 0, -20, 10]
arr = list(filter(func, arr))
print(arr)  # Результат: [2, 4, 0, 10)
```

Приклад 33. Вилучити елементи списку з використанням генератора списків

```
def func(elem):
    return elem >= 0

arr = [-1, 2, -3, 4, 0, -20, 10)

arr = [ i for i in arr if func(i) ]

print(arr)
```

Результат: [2, 4, 0, 10]

Функція `reduce()`

Функція **`reduce()`** з модуля `functools` застосовує зазначену функцію до пар елементів і накопичує результат. Функція має наступний формат:

reduce(<Функція>, <Послідовність>[, <Початкове значення>])

У функцію зворотного виклику як параметри передаються два елементи:

- перший елемент буде містити результат попередніх обчислень,

- другий елемент – значення поточного елемента.

Приклад 34. Одержимо суму всіх елементів списку

```
import functools
def func (x, y):
    print("({0}, {1})".format(x, y), end=" ")
    return x + y
arr = [1, 2, 3, 4, 5]
suma = functools.reduce(func, arr)
# Послідовність: (1, 2) (3, 3) (6, 4) (10, 5)
print('вариант1',suma) # Результат виконання: 15
suma = functools.reduce(func, arr, 10)
# Послідовність:(10,1) (11,2) (13,3) (16,4) (20,5)
print('вариант2',suma) # Результат виконання: 25
suma = functools.reduce(func, [], 10)
print('вариант3',suma) # Результат виконання: 10

del functools
suma = functools.reduce(func, [1,1], 10)
print('вариант4',suma)
# Помилка через вивантаження модуля functools
```

Додавання й видалення елементів списку

Для додавання й видалення елементів списку використовуються наступні методи:

`append (<Об'єкт>)` – додає один об'єкт у кінець списку. Метод змінює поточний список і нічого не повертає.

Приклад 35.

```
>>> arr = [1, 2, 3]
>>> arr.append(4); arr
[1, 2, 3, 4]
>>> arr. append([5, 6])
>>> arr
[1, 2, 3, 4, [5, 6]]
>>> arr. append ( (7, 8) )
arr
[1, 2, 3, 4, [5, 6], (7, 8)]
```

`extend(<Послідовність>)` – додає елементи послідовності в кінець списку. Метод змінює поточний список і нічого не повертає.

Приклад 36.

```
>>> arr = [1, 2, 3]
>>> arr.extend([4, 5, 6]) # Додаємо список
>>> arr. extend ((7, 8, 9)) # Додаємо кортеж
>>> arr.extend("abc") # Додаємо букви з рядка
>>> arr
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 'a'' 'b', 'c']
```

Оператор += – додає елементи за допомогою операції конкатенації

Приклад 37.

```
>>> arr = [1, 2, 3]
>>> arr + [4, 5, 6] # Повертає новий список
[1, 2, 3, 4, 5, 6]
>>> arr = [1, 2, 3]
>>> arr += [4, 5, 6]
>>> arr   # Змінює поточний список
[1, 2, 3, 4, 5, 6]
```

Крім того, можна скористатися операцією присвоювання значення зрізу:

Приклад 38.

```
>>> arr = [1, 2, 3]
>>> arr[len(arr):] = [4, 5, 6]# Змінює список
>>> arr
[1, 2, 3, 4, 5, 6]
```

`insert (<Індекс>, <Об'єкт>)` – додає один об'єкт у зазначену позицію. Інші елементи зміщаються. Метод змінює поточний список і нічого не повертає.

Приклад 39.

```
>>> arr = [1, 2, 3]

>>> arr.insert(0, 0); arr # Вставляємо 0 у
початок списку
[0, 1, 2, 3]

>>> arr.insert(-1, 20); arr # Можна вказати
від’ємні
[0, 1, 2, 20, 3]

>>> arr.insert(2, 100); arr # Вставляємо 100 у
позицію 2
[0, 1, 100, 2, 20, 3]
```

```
>>> arr.insert(10, [ 4, 5]); arr
# Додаємо список
[0, 1, 100, 2, 20, 3, [4, 5]]
```

Метод `insert ()` дозволяє додати тільки один об'єкт. Щоб додати кілька об'єктів, можна скористатися операцією присвоювання значення зрізу.

Додамо кілька елементів у початок списку:

Приклад 40.

```
>>> arr = [1, 2, 3]
>>> arr[:0] = [-2, -1, 0]
>>> arr
[-2, -1, 0, 1, 2, 3]
```

`pop([<Індекс>])` – видаляє елемент, розташований по зазначеному індексу, і повертає його. Якщо індекс не зазначений, то видаляє й повертає останній елемент списку. Якщо елемента із зазначеним індексом немає, або список порожній, виконується виключення `Indexerror`.

Приклад 41.

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr.pop() # Видаляємо останній елемент
списку
5
>>> arr # Список змінився
[1, 2, 3, 4]
>>> arr.pop(0) # Видаляємо перший елемент списку
1
>>>arr # Список змінився
[2, 3, 4]
```

Вилучити елемент списку дозволяє також оператор `del`:

Приклад 42.

```
>>> arr = [1, 2, 3, 4, 5]
>>> del arr[4]; arr # Видаляємо останній елемент списку
[1, 2, 3, 4]
>>> del arr[:2]; arr # Видаляємо перший і другий елементи
[3, 4]
```

`remove (<Значення>)` – видаляє перший елемент, утримуючий зазначене значення. Якщо елемент не знайдений, виконується виключення `Valueerror`. Метод змінює поточний список і нічого не повертає.

Приклад 43.

```
>>> arr = [ 1, 2, 3, 1, 1 ]
>>> arr.remove(1) # Видаляє тільки першу одиничку
```

```
>>> arr
[2, 3, 1, 1]
>>> arr.remove(S) # Такого елемента немає
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Nameerror: name 'S' is not defined
>>> arr.remove(3) Видаляє елемент 3
>>> arr
[2, 1, 1]
>>> arr.remove(1) Видаляє елемент одиничку
>>> arr
[2, 1]
>>> arr.remove(5) # Такого елемента немає
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Valueerror: list.remove(x): x not in list
```

`clear()`- видаляє всі елементи списку, очищаючи його. Жодного результату при цьому не повертається. Підтримка цього методу з'явилася в Python3.3.

Приклад 44

```
>>> arr = [1, 2, 3, 1, 1]
>>> arr.clear()
>>> arr
[]
```

Технологія вилучення повторень

Якщо необхідно вилучити всі повторювані елементи списку, то можна перетворити список у множину, а потім множину знов перетворити в список.

Список повинен містити тільки незмінювані об'єкти (наприклад, числа, рядки або кортежі). В протилежному випадку виконується виключення `Typeerror`.

Приклад 45

```
>>> arr = [ 1, 2, 3, 1, 1, 2, 2, 3, 3]
>>> s = set(arr) # Перетворимо список у множину
>>> s
{1, 2, 3}
>>> arr = list(s) # Перетворимо множину в список
>>> arr # Усі повтори були вилучені
[1, 2, 3]
```