Лекція 11

Списки, кортежі, множини і діапазони в мові Python

Алгоритм вибору варіанту контрольної роботи №1

1. Взяти свій номер у списку групи.
2. Розділити свій номер на 5.
3. Одержаний залишок від ділення буде номером вашого варіанту.
4. Навести обчислення в контрольній роботі.

Наприклад.

Номер у списку: 5. Тоді 5:5 = 1 залишок: 0

Номер у списку: 1. Тоді 1:5 = 0 залишок: 1

Номер у списку: 12. Тоді 12:5 = 2 залишок: 2

Номер у списку: 28. Тоді 28:5 = 5 залишок: 3

Номер у списку: 19. Тоді 19:5 =3 залишок: 4

I. Яким буде вивід інтерпретатора?

0.
```
>>> print("Life\\nis\nice")
```

1.
```
>>> print("I have\
A dream")
```

2.
```
>>> print("Слово було\\")
```

3.
```
>>> print ("""Коник
Стрибунець""")
```

4.
```
>>> print(r"Mice\nice")
```

II. Дано рядок

```
>>> f = "Як тебе не любити, Києве мій!"
```

Записати інструкцію інтерпретатору, щоб

0. Створити копію рядка

1. Вивести символи у зворотному порядку

2. Замінити перший символ у рядку на символ «О»

3. Вилучити останній символ

4. Одержати останній символ:

III. Дано рядок

```
>>>f = "Грає море зелене, Тихий день догора"
```

Записати інструкцію інтерпретатору, щоб

0. замінити всі символи рядка відповідними великими буквами:

1. замінити всі символи рядка відповідними малими літерами:

2. замінити всі малі символи відповідними великими буквами, а всі великі символи – малими:

3. змінити першу букву кожного слова на велику

4. знайти підрядок « море» в даному рядку.

IV. Яким буде результат роботи інтерпретатора ?.

0.
```
>>>bytes("\ufffds\ufffd","cp1251","replace")?
```

1.
```
>>> b = bytes ( "fiot", "cp1251"); b[:-1]
```

2.
```
>>>len(bytes('НТУУ"КПІ"', "utf-8"))
```

3.
```
>>> "'%3d' - '%-3d'" % (3, 3)
```

4.
```
>>>"{name}-{age}".format(age="18", name ="Ivan")
```

Питання оцінюється в 1 бонусний бал!!!!

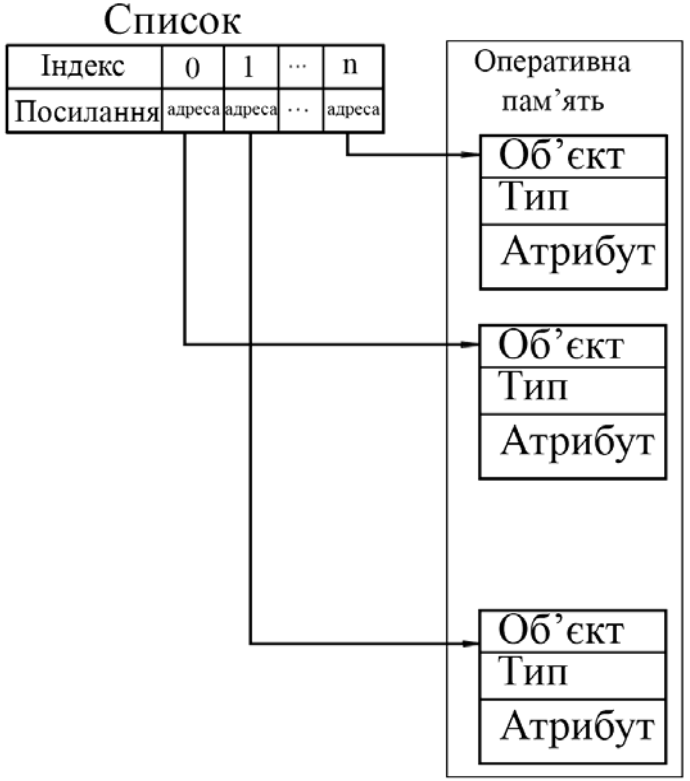
Чи будуть однаковими результати роботи інтерпретатора для першого і другого варіанту?

Варіант 1	Варіант2
`>>> -2**0.5`	`>>> x=-2` `>>> x**0.5`

Відповідь повинна бути обґрунтованою

Списки, кортежі, множини й діапазони – це індексовані послідовності об'єктів.

Кожний елемент таких послідовностей містить лише посилання на об'єкт - тому вони можуть містити об'єкти довільного типу даних і мати необмежений ступінь вкладеності.



Властивості послідовностей

1. Позицію елемента в наборі задають індексом.

2. Нумерація елементів починається з 0.

3. Списки й кортежі є просто впорядкованими послідовностями елементів.

Як і всі послідовності, вони підтримують:
- доступ до елемента по індексу,
- одержання зрізу,
- конкатенацію (оператор +),
- повторення (оператор *),
- перевірку на входження (оператор `in`)
- перевірку на невходження (оператор `not in`).

Списки

1. Списки є змінюваними типами даних.

Це означає, що ми можемо не тільки одержати елемент по індексу, але й змінити його:

Приклад 1.

```
>>> arr = [1, 2, 3] # Створюємо список

>>> arr[0] # Одержуємо елемент по індексу
1

>>> arr[0] = 50 # Змінюємо елемент по індексу

>>> arr
[50, 2, 3]
```

Кортежі

2. Кортежі є незмінюваними типами даних.
Іншими словами, можна одержати елемент по індексу, але змінити його не можна:

Приклад 2.

```
>>> 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+
```

Множини

3. Множини можуть бути як змінюваними, так і незмінюваними. Їхня основна відмінність від щойно розглянутих типів даних – зберігання лише унікальних значень (неунікальні значення автоматично відкидаються).

Приклад 3.

```
>>> set([0, 1, 1, 2, 3, 3, 4])
{0, 1, 2, 3, 4}

>>> set(["a", "b", "c", "c", "d"])
{'a', 'd', 'b', 'c'}

>>> set(["string", 1, "string", 3, 3])
{'string', 1, 3}
```

Діапазони

4. Діапазони є наборами чисел, сформованими на основі заданих
- початкового значення величини,
- кінцевого значення,
- величини кроку між числами.

Їхня найважливіша перевага перед усіма іншими наборами об'єктів – невеликий обсяг оперативної пам'яті для зберігання

Приклад 4.

```
r = range(0, 101, 10)
for i in r: print(i, end = " ")
```

Результат роботи програми:

```
0 10 20 30 40 50 60 70 80 90 100
```

Створення списку

Створити список можна такими способами:

1. За допомогою функції `list([<Послідовність>])`.

Функція дозволяє перетворити будь-яку послідовність у список. Якщо параметр не зазначений, то створюється порожній список.

Приклад 5.

```
>>> list() # Створюємо порожній список
[]
>>> list("String") # Перетворимо рядок у список
['S', 't', 'r', 'i', 'n', 'g')
>>> list ((1, 2, 3, 4, 5)) # Перетворимо кортеж у
список
[1, 2, 3, 4, 5]
```

```
s = {"a", "string", "рядок", 12, 45.123, True}
b = list(s)
print(b)
[True, 'string', 12, 45.123, 'рядок', 'a']
```

2. Перелічивши всі елементи списку усередині квадратних дужок:

Приклад 6.

```
>>> arr = [1, "str", 3, "4"]
>>> arr
[ 1, 'str', 3, '4']

>>> lmy = ["Петренко", "Петро", 22, "року"]
>>> lmy
['Петренко', 'Петро', 22, 'року']

>>> nmy = [12.2, 11.234567, 23e-12]
>>> nmy
[12.2, 11.234567, 2.3e-11]
```

3. Застосувавши метод append() для заповнення списку поелементно:

Приклад 7.

```
>>> arr = [] # Створюємо порожній список

>>> arr.append(1)#додаємо елемент 1(індекс 0)

>>> arr.append("str") # Додаємо елемент "str"
(індекс 1)

>>> arr

[1, 'str']
```

Відмінність від PHP

У деяких мовах програмування (наприклад, у PHP) можна додати елемент, указавши порожні квадратні дужки або індекс більший, ніж останній індекс. У мові Python усі ці способи призведуть до помилки:

```
>>> arr = []
>>> arr[] = 10          #arr+=[10], arr.append(1)
  File "<input>", line 1
    arr[] = 10
        ^
Syntaxerror: invalid syntax

>>> arr[0] = 10 # arr=[10]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Indexerror: list assignment index out of range
```

Особливості створення списку

1. При створенні списку в змінній зберігається посилання на об'єкт, а не сам об'єкт.
2. Це обов'язково слід враховувати при груповому присвоюванні.
3. Групове присвоювання можна використовувати для чисел і рядків, але для списків цього робити не можна.

Приклад 8.

```
>>> x = y = [1, 2] # Нібито створили два об'єкти
>>> x, y
([1, 2], [1, 2])
```

У цьому прикладі ми створили список з двох елементів і присвоїли значення змінним `x` и `y`.

Тепер спробуємо змінити значення в змінній `y`:

```
>>> y[1] = 100  # Змінюємо другий елемент
>>> x, y       # Змінилося значення відразу у двох змінних
([1, 100], [1, 100])
```

```
>>> x = y = [1, 2]
>>> x is y
True
```

Як видно з прикладу, зміна значення в змінній `y` привела також до зміни значення в змінній `x`.

Таким чином, обидві змінні посилаються на той самий об'єкт, а не на два різних об'єкти.

Щоб одержати два об'єкти, необхідно робити роздільне присвоювання:

Приклад 9.

```
>>> x, y = [1, 2], [1, 2]
>>> y[1] = 100 # Змінюємо другий елемент
>>> x, y
([1, 2], [1, 100])
>>> x,y,z=2,2,2 # Слід відрізняти від кеширування
>>> x is y
True
>>> y=3
>>> x is y
False
```

Особливості створення списку з використанням оператора *

Точно така ж ситуація виникає при використанні оператора повторення *.
Наприклад, у наступній інструкції проводиться спроба створення двох вкладених списків за допомогою оператора *:

Приклад 10.

```
>>> arr = [ [] ] * 2 # Нібито створили два вкладені списки
>>> arr
[[], []]
>>> arr[0].append(5) # Додаємо елемент
>>> arr # Змінилися два елементи
[[5], [5]]
```

Створення вкладених списків

Створювати вкладені списки слід за допомогою методу `append()` всередині циклу:

Приклад 11.

```
>>> arr = []

>>> for i in range(2): arr.append([])

>>> arr
[[], []]

>>> arr[0].append(5)

>>> arr
[[5], []]
```

Створення списків за допомогою генераторів

Генератор списків – спосіб побудувати новий список, застосовуючи вираз до кожного елемента послідовності.

Генератори списків дуже схожі на цикл for.

Приклад 12.

```
>>> arr = [ [] for i in range(2) ]
>>> arr
[[], []]
>>> arr[1].append(2)
>>> arr
[[1], [2]]
>>> arr[0].append(3)
>>> arr
[[1, 3], [2]]
>>> arr[0].append('str')
>>> arr
[[1, 3, 'str'], [2]]
```

Перевірка змінних

1. Перевірити, чи посилаються дві змінні на той самий об'єкт, дозволяє оператор `is`.
2. Якщо змінні посилаються на той самий об'єкт, то оператор `is` повертає значення `True`:

Приклад 13.

```
>>> x = y = [1, 2] # Неправильно
>>> x is y # Змінні містять посилання на той самий список
True

>>> x, y = [1, 2], [1, 2] # Правильно
>>> x is y  # Це різні об'єкти
False
```

Створення копії списку

Існують три способи створити копію списку:

1. За допомогою функції `list()`.
2. З застосуванням операції добування зрізу.
3. З застосуванням методу `copy()`.

Приклад 14. Створення копії за допомогою `list()`

```
>>> x = [1, 2, 3, 4, 5] # Створили списки
>>> y = ["a", "b", "c", "d"]
>>> # створюємо копію списку за допомогою list
>>> z = list(x)
>>> f = list(y)
>>> z,f
([1, 2, 3, 4, 5], ['a', 'b', 'c', 'd'])
>>> z is x
False
>>> f is y
False
```

Створення копії за допомогою добування зрізу

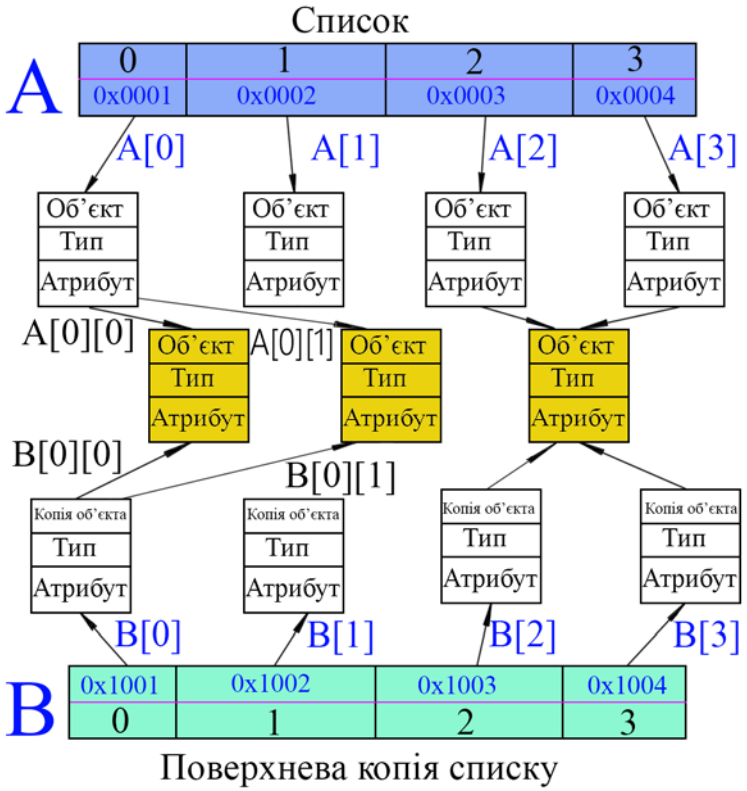
```
>>> x = [1, 2, 3, 4, 5] # Створили списки

>>> y = ["a", "b", "c", "d"]

>>> z= x[:]

>>> f= y[:]

>>> z,f

([1, 2, 3, 4, 5], ['a', 'b', 'c', 'd'])

>>> f is y
False
>>> z is x
False
```

Створення копії списку викликом методу `copy()` :

```
>>> x = ["2014","2015","2016","2017"]
>>> y= x.copy()
>>> y
['2014', '2015', '2016', '2017']

>>> x is y # Оператор показує, що це різні об'єкти
False

>>> y[1] = "2018" # Змінюємо другий елемент

>>> x,y # Змінився тільки список у змінній y

(['2014','2015','2016','2017'],
['2014','2018','2016','2017'])
```

Поверхнева копія списку

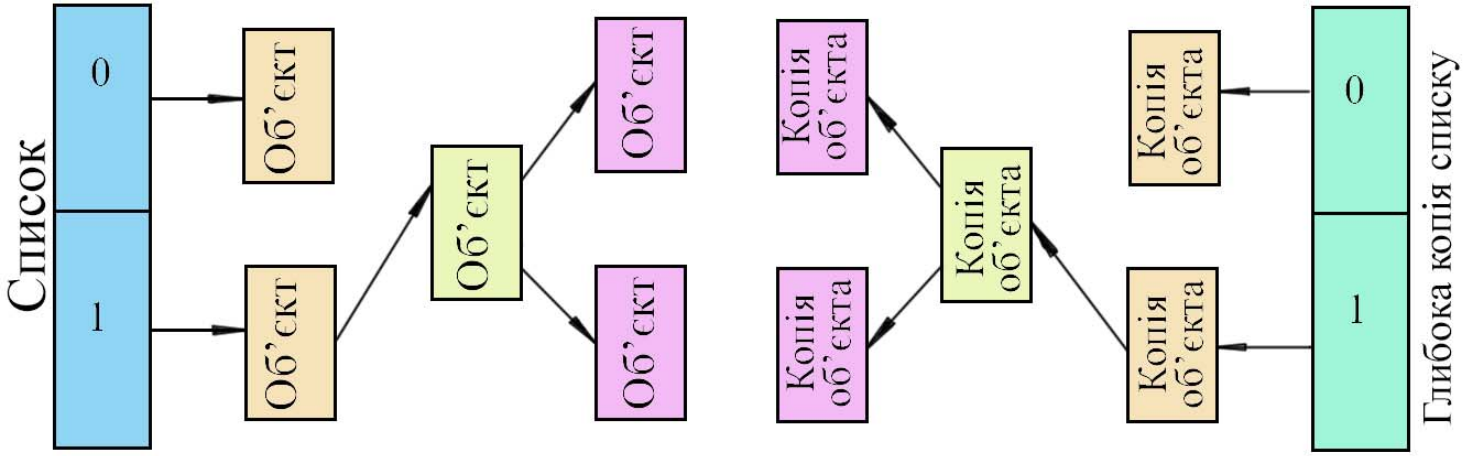
1. Операція присвоювання не копіює об'єкт, вона лише створює посилання на об'єкт.
2. Розглянуті методи створення копії списку створюють поверхневу копію.



Означення поверхневої та глибокої копії

Поверхнева копія створює новий складений об'єкт, і потім (у міру можливості) вставляє в нього посилання на об'єкти, що містяться в оригіналі.

Глибока копія створює новий складений об'єкт, і потім рекурсивно вставляє в нього копії об'єктів, що містяться в оригіналі.



Поверхнева копія

Приклад 15.

```
>>> x = [1, [2, 3, 4, 5]]# Створили вкладений список
>>> y = list(x)              # Зробили копію списку
>>> x is y                   # Різні об'єкти
False
>>> y[1] [1] =100  # Змінюємо елемент
>>> y[0]=200
>>> x,y # Зміна торкнулася змінної x!!!
([1, [2, 100, 4, 5]], [200, [2, 100, 4, 5]])
```

У прикладі список `y` є поверхневою копією списку `x`. **Доказ:**

```
>>> x is y
False
```

Отже, на верхньому рівні `x` і `y` – різні об'єкти.

```
>>> x,y # Зміна торкнулася змінної x!!!
([1, [2, 100, 4, 5]], [200, [2, 100, 4, 5]])
```

Отже, `x` і `y` містять однакові посилання на вкладені об'єкти.

Глибока копія

Щоб одержати глибоку копію списку, слід скористатися функцією `deepcopy()` з модуля `copy()` .

Приклад 16

```
>>> import copy # Підключаємо модуль copy
>>> x = [1, [2, 3, 4, 5]]
>>> y = copy.deepcopy(x)#створюємо глибоку копію
>>> y[1] [1] = 100   # Змінюємо другий елемент
>>> x,y  # Змінився тільки список в y
([1, [2, 3, 4, 5]], [1, [2, 100, 4, 5]])
```

Розглянемо приклад, в якому два елементи посилаються на один об'єкт.

Функція `deepcopy()` рекурсивно створює копію кожного вкладеного об'єкта, при цьому зберігаючи внутрішню структуру списку.

Приклад 17

```
>>> import copy    # Підключаємо модуль copy
>>> x = [1, 2]
>>> y = [x, x] # два елементи посилаються на один об'єкт
>>> y
[[1, 2], [1, 2]]
>>> z = copy.deepcopy(y) # Зробили копію списку
>>> z[0] is x, z[1] is x, z[0] is z[1]
(False, False, True)
>>> z [0] [0] = 300  # Змінили один елемент
>>>  z  #  Значення  змінилося  відразу  у  двох
елементах!
[[300, 2], [300, 2]]
>>> x   # Початковий список не змінився
[1, 2]
```

Операції над списками

1. Доступ до елементів списку здійснюється за допомогою квадратних дужок.

2. У квадратних дужках вказується індекс елемента.

3. Нумерація елементів списку починається з нуля.

Приклад 18. Вивід елементів списку:

```
>>> arr = [1, "str", 4.1, "5"]

>>> arr[0], arr[1], arr[2], arr[3]

(1, 'str', 4.1, '5')

>> arr[0]
1
```

Позиційне присвоювання для списків

Особливість позиційного присвоювання:
Кількість елементів праворуч і ліворуч від оператора = повинні збігатися, інакше буде виведене повідомлення про помилку:

Приклад 19

```
>>> x, y, z = [1, 2, 3] # Позиційне присвоювання
>>> x, y, z
(1, 2, 3)
>>> x, y= [1, 2, 3] # Кількість елементів повинна збігатися

Traceback (most recent call last):
  File "<input>", line 1, in <module>
Valueerror: too many values to unpack (expected
2)
```

Зірочка в позиційному присвоюванні

1. В Python 3 при позиційному присвоюванні перед однією зі змінних ліворуч від оператора = можна вказати зірочку (*).

2. У цій змінній буде зберігатися список, що містить «зайві» елементи. Якщо таких елементів немає, то список буде порожнім:

Приклад 20

```
>>> x, y, *z = [1, 2, 3]; x, y, z
(1, 2, [3])

>>> x, y, *z = [1, 2, 3, 4, 5]; x, y, z
(1, 2, [3, 4, 5])
>>> x, y, *z = [1, 2]; x, y, z
(1, 2, [])
```

```
>>> *x, y, z = [1, 2]; x, y, z
([], 1, 2)

>>> x, *y, z = [1, 2, 3, 4, 5]; x, y, z
(1, [2, 3, 4], 5)

>>> *z, = [1, 2, 3, 4, 5]; z
[1, 2, 3, 4, 5]
```

Зірочка повинна бути тільки одна.

```
>>> x, *y, *z = [1, 2, 3, 4, 5]; x, y, z

  File "<input>", line 1
Syntaxerror:   two   starred   expressions   in
assignment
```

Використання індексів у списках

Список `mylist`				
Елемент	1	2	3	4
Індекс	0	1	2	3

Оскільки нумерація елементів списку починається з 0, індекс останнього елемента буде на одиницю меншим від кількості елементів.

Функція `len()` – повертає кількість елементів списку

Приклад 21

```
>>> arr = [1, 2, 3, 4, 5, 6]
>>> len(arr) # Одержуємо кількість елементів
6
>>> arr[len(arr)-1] # Одержуємо останній елемент
6
```

Некоректний індекс

Якщо елемент, відповідний до зазначеного індексу, відсутній у списку, то виконується виключення `Indexerror`:

Приклад 22

```
>>> arr = [1, 2, 3, 4, 5, 6]
>>> arr[6]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Indexerror: list index out of range

arr = [1, 'str', 4.1, '5']
>>> arr[5]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Indexerror: list index out of range
```

Від'ємний індекс

1. Як індекс можна вказати від'ємне значення.
2. Зсув буде відлічуватися від кінця списку.

Приклад 23

```
>>> arr = [1, 2, 3, 4, 5, 6]
>>> arr[-6], arr[-1] #перший і останній елемент
(1, 6)
```

Оскільки списки є змінюваними типами даних, то можна змінити елемент по індексу:

Приклад 24

```
>>> arr = [1, 2, 3, 4, 5, 6]
>>> arr[3] = 100 #змінюємо 4-й елемент
>>> arr
[1, 2, 3, 100, 5, 6]
>>> arr[-6] = 50 # Змінюємо (-6)-й елемент
>>> arr
[50, 2, 3, 100, 5, 6]
```

Операція зрізу

1. Списки підтримують операцію добування зрізу
2. Операція повертає зазначений фрагмент списку.

Формат операції:

```
[<Початок>:<Кінець>:<Крок>]
```

Усі параметри не є обов'язковими.

1. Якщо параметр <Початок> не зазначений, то використовується значення 0.
2. Якщо параметр <Кінець> не зазначений, то повертається фрагмент до кінця списку. Слід також відмітити, що елемент з індексом, зазначеним в цьому параметрі, не входить у фрагмент, що повертається.
3. Якщо параметр <Крок> не зазначений, то використовується значення 1.
4. Як значення параметрів можна вказати від'ємні значення.

Застосування операції зрізу

Спочатку одержимо поверхневу копію списку:

Приклад 25

```
>>> arr = [1, 2, 3, 4, 5, 6]
>>> m = arr[1:6:2]
>>>  #елемент   з індексом 6 не входить у фрагмент, що повертається
>>> m
[2, 4, 6]
>>> m is arr
False
>>>#якщо потрібно скопіювати всі елементи списку поверхнево
>>> s = arr[:];
>>> s
[1, 2, 3, 4, 5, 6]
>>> s is arr
False
```

Приклад 26

Виведемо список у зворотному порядку

```
>>> arr = [1, 2, 3, 4, 5, 6]
>>> arr[::-1# крок -1
[6, 5, 4, 3, 2, 1]
```

Виведемо список без першого й останнього елементів

```
>>> arr[1:] # Вивід списку без першого елемента
[2, 3, 4, 5, 6]
>>> arr[:-1]# Вивід без останнього елемента
[1, 2, 3, 4, 5]
>>> arr[0:2] # Одержимо перші два елементи
[1, 2] # Символ з індексом 2 не в діапазоні
>>> arr[-1:] # Останній елемент списку
[6]
>>> arr[1:4]# Повертаються елементи з індексами 1, 2 і 3
[2, 3, 4]
```

Зміна й видалення фрагмента списку

1. За допомогою зрізу можна змінити фрагмент списку.
2. Якщо зрізу присвоїти порожній список, то елементи, що потрапили в зріз, будуть вилучені:

Приклад 27

```
>>> arr = [1, 2, 3, 4, 5, 6]

>>> arr[1:3] = [8,7]# Змінюємо елементи з індексами 1 і 2
>>> arr
[1, 8, 7, 4, 5, 6]

>>> arr[1:3]=[ ]#Видаляємо елементи з індексами 1 і 2
>>> arr
[ 1, 4, 5, 6]
```

Конкатенація списків

З'єднати два списки в один список дозволяє оператор +. Результатом об'єднання буде новий список:

Приклад 28

```
>>> arr1 = [1, 2, 3, 4, 5, 6]
>>> arr2 = [7, 8, 9]
>>> sum = arr1 + arr2
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Замість оператора + можна використовувати оператор +=. Слід враховувати, що в цьому випадку елементи додаються в поточний список:

Приклад 29

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

Операції повторення (*) і перевірки на входження (in, not in) для списків

1. Повторити список зазначену кількість разів можна за допомогою оператора *.
2. Виконати перевірку на входження елемента в список дозволяє оператор in :

Приклад 30

```
>>> ["a", "b", "c"]*2 # Операція повторення
['a', 'b', 'c', 'a', 'b', 'c']
>>> ["a", "b", "c"]*3  # Операція повторення
['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']

>>> # Перевірка на входження
>>> "a" in ["a", "b", "c"], "d" in ["a", "b", "c"]
(True, False)
```

Багатовимірні списки

Будь-який елемент списку може містити об'єкт довільного типу.

Наприклад, елемент списку може бути:

- числом,
- рядком,
- списком,
- кортежом,
-словником і т. д.

Створити вкладений список можна, наприклад, так:

Приклад 30

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

Вираз всередині дужок може розташовуватися на декількох рядках. Отже, попередній приклад можна записати інакше:

Приклад 31

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

Щоб одержати значення елемента у вкладеному списку, слід указати два індекси:

Приклад 32

```
>>> nested[0]
[1, 2, 3]
>>> nested[0][0]
1
```

Елементи вкладеного списку також можуть мати елементи довільного типу. Кількість вкладень не обмежена.

Можна створити об'єкт будь-якого ступеня складності.

Для доступу до елементів вказується кілька індексів підряд.

Приклад 32

Елементи – списки.

```
>>> nested = [
                [["a", "b","A"],1],
                [["c", "d","B"],2],
                [["e", "f","C"],3],
              ]
>>> nested[0][1], nested[1][1], nested[2][1]
(1, 2, 3)
>>> nested[0][0][0], nested[1][0][1],
nested[2][0][2]
('a', 'd', 'C')
```

Елементи – кортежі.

```
>>> nested = [
                [("a", "b","A"),1],
                [("c", "d","B"),2],
                [("e", "f","C"),3],
```

```
                ]
>>> nested[0][0][0], nested[1][0][1],
nested[2][0][2]
('a', 'd', 'C')
```

Елементи – словники.

```
>>> nestdic = [1, "a", {"b":10,"c":["d", 100]}]
>>> nestdic[0]
1
>>> nestdic[1]
'a'
>>> nestdic[2]
{'c': ['d', 100], 'b': 10}
>>> nestdic[2]["c"]
['d', 100]
>>> nestdic[2]["c"][0]
'd'
```