**Списки**

**Списки, на відміну від рядків, є об'єктами, що змінюються : можна окремому елементу списку присвоїти нове значення. Але можна міняти і повністю зрізи. Наприклад:**

**A = [1, 2, 3, 4, 5]**

**A[2:4] = [7, 8, 9]**

**A = [1, 2, 7, 8, 9, 5]**

**Зверніть увагу, що A[i] - це елемент списку, а не зріз!**

**Списки в Python - впорядковані колекції об'єктів довільних типів, що змінюються (майже як масив, але типи можуть відрізнятися).**

**Введення/створення**

1. **Приклади наданих списків**

**Primes = [2, 3, 5, 7, 11, 13]**

**Rainbow = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']**

1. **Функція list** - можна обробити будь-який об'єкт, що ітерується (наприклад, [рядок](https://pythonworld.ru/tipy-dannyx-v-python/stroki-literaly-strok.html) ) вбудованою функцією **list**

**>>> list('список')**

**['c', 'п', 'и', 'c', 'о', 'к']**

**a = list('список')**

**print(a)**

1. **Користувальницький ввід:**

**a = []** # Заводимо порожній список

**n = int(input())** # зчитуємо кількість елементів у списку

**for i in range(n):**

**new\_element = int(input())** # зчитуємо черговий елемент

**a.append(new\_element)** # додаємо його до списку

останні два рядки можна було замінити одним:

# a.append(int(input()))

**print(a)**

1. **Цикл for :**   
   За індексами –

**a = [0] \* int( input())**

**for i in range(len(a)):**

**a[i] = int(input())**

1. **Методи split та join:**

Елементи списку можуть вводитися по одному у рядку

# на вхід подається рядок

# 1 2 3

**s = input()** # s == '1 2 3'

**a = s.split()** # a == ['1', '2', '3']

елементи списку по одному перетворити на числа

**a = input().split()**

**for i in range(len(a)):**

**a[i] = int(a[i])**

або генератором:

**a = [int(s) for s in input().split()]**

Зворотня операція:

**a = ['red', 'green', 'blue']**

**print(' '.join(a))**

або генератором :

**a = [1, 2, 3]**

**print(' '.join([str(i) for i in a]))**

**a = [1, 2, 3]**

**aas = ' '.join ([str(i) for i in a])**

**print(aas)**

**print(' '.join([str(i) for i in a]))**

**dd = aas.split('')**

**print(dd)**

1. **Генератори списків:**

**[Вираз for змінна in послідовність]**

**n = 5**

**a = [i \*\* 2 for i in range(n)]**

**>>> [0, 1, 4, 9, 16]**

**[вираз for змінна in послідовність if умова]**

**n = 5**

**a = [i \*\* 2 for i in range(n) if i % 2 == 0]**

**>>> [0, 4, 16]**

**[n for n in range(1, 10000) if n % 2 == 0]**

**[(x, y) for x in range(1,10) for y in range(1,10) if x % y == 0]**

Ось так можна отримати список, заповнений випадковими числами від 1 до 9:

**from random import randrange**

**n = 10**

**a = [randrange(1, 10) for i in range(n)]**

**a = [randrange(-10,11) for i in range(randrange(6,11))]**

А в цьому прикладі список складатиметься з рядків, зчитаних зі стандартного введення: спочатку потрібно ввести число елементів списку (це значення буде використано як аргумент функції range), потім - задану кількість рядків:

**a = [input() for i in range(int(input()))]**

**Більше складний випадок :**

**c = [c + d for c in 'list' if c! = ' i ' for d in 'spam' if d != 'a']**

**['ls', ' lp ', ' lm ', 'ss', ' sp ', ' sm ', ' ts ', ' tp ','tm']**

**У мові Python є і вирази-генератори, які мають схожий зі списковими включеннями синтаксис, але повертають** [**ітератор**](https://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80) [**[11]**](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BA%D0%BE%D0%B2%D0%BE%D0%B5_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B5#cite_note-_282896e39f5fe0c9-11) **. Сума парних чисел із попереднього прикладу:**

**sum((n for n in range(1, 10000) if n % 2 == 0))**

**Операції зі списками**

1. **Створення копії списку**
2. L1 = L2[:] – створення другої копії списку. Тут створюється друга копія об'єкта.
3. L1 = list(L2) - також створення другої копії списку.
4. L1 = L2 — створення другого посилання, а не копії. Третій варіант показує, що створюються два посилання на той самий об'єкт, а не дві копії.
5. **Додавання або конкатенація списків:**

L1 + L2

1. **Множення, або повтор списків:**

L1\*2

1. **Ітерація списків:**

* проста ітерація списку:

| 1 | for x in L: |
| --- | --- |

* сортована ітерація:

| 1 | for x in sorted(L): |
| --- | --- |

* унікальна ітерація:

| 1 | for x in set(L): |
| --- | --- |

* ітерація у зворотному порядку:

| 1 | for x in reversed(L): |
| --- | --- |

* виключна ітерація - наприклад, вивести елементи 1-го списку, яких немає у 2-му списку:

| 1 | for item in set(L).difference(L2) |
| --- | --- |

1. **Операція Sequence unpacking — надання списку змінних списку значень:**

| 1 | a, b = [1,2] |
| --- | --- |

1. **Перевірити, чи міститься елемент у списку. Повертає True або False**

**x in A**

1. **x not in A Те ж саме, що not (x in A)**
2. **Найменший елемент списку**

**min(A)**

1. **Найбільший елемент списку**

**max(A)**

1. **Індекс першого входження елемента x у список, за його відсутності генерує виключення ValueError**

**A.index(x)**

1. **Кількість входжень елемента x до списку**

**A.count(x)**

**Функції та методи списків**

**Таблиця "методи списків"**

| **Метод** | **Що робить** |
| --- | --- |
| **list.append** (x) | Додає елемент до кінця списку |
| **list.extend** (L) | Розширює список list, додаючи до кінця всі елементи списку L |
| **list.insert** (i, x) | Вставляє на i-ий елемент значення x |
| **list.remove** (x) | Видаляє перший елемент у списку, який має значення x. ValueError, якщо такого елемента не існує |
| **List. del (x)** | Вилучення за індексом |
| **list.pop** ([i]) | Видаляє і-ий елемент і повертає його. Якщо індекс не вказано, видаляється останній елемент |
| **list.index** (x, start, end) | Повертає положення першого елемента зі значенням x (при цьому пошук ведеться від start до end) |
| **list.count**(x) | Повертає кількість елементів зі значенням x |
| **list.sort** ([key=функція]) | Сортує список на основі функції |
| **list.reverse** () | Розвертає список |
| **list.copy** () | Поверхнева копія списку |
| **list.clear** () | Очищає список |

**При видаленні ( del , pop , remove ) потрібно пам'ятати про те, що не можна одночасно робити ітерацію за списком - наслідки будуть непередбачувані.**

**Вбудовані функції filter(), map(), zip(), reduce()**

**filter( function, sequence)** повертає послідовність, що складається з елементів послідовності sequence, для яких function(item) є істиною. Функція застосовується до кожного елемента послідовності. Приклад: визначимо прості числа в діапазоні до 100:

**def f(x):**

**for y in range(2, x):**

**if x%y==0: return 0**

**return 1**

**print (\* filter(f, range(2, 100)))**

**>>> [2, 3, 5, 7, 11, 13, 17, 19, 23, ... , 59, 61, 67, 71, 73, 79, 83, 89, 97]**

**numbers = [10, 4, 2, -1, 6]**

**>>> filter( lambda x: x < 5, numbers) *# В результат потрапляють тільки ті елементи x, для яких x < 5 істинно***

**[4, 2, -1]**

**Те саме за допомогою спискових виразів:**

**numbers = [10, 4, 2, -1, 6]**

**>>> [x for x in numbers if x < 5]**

**[4, 2, -1]**

**map( function, sequence)** повертає список значень, отриманих застосуванням функції function до елементів однієї чи кількох послідовностей. Наприклад, створимо список кубів натуральних чисел від 1 до 10:

**def cube (x): return x \* x \* x**

**print (\* map( cube, range(1, 11)))**

**numbers = map(lambda x: x\*\*2, range(10))**

**print(\*numbers)**

**ось так – генератором:**

**l = [x\*\*2 for x in range(1,10)]**

**print(\*l)**

**old\_list = [1, 2, 3, 4]**

**new\_list = map(lambda x: x+1, old\_list)**

**print(\*new\_list)**

**def upper(s):**

**return s.upper()**

**print(list(map(upper, ['sentence', 'fragment']))) або**

**print([upper(s) for s in ['sentence', 'fragment']])**

**func = lambda el1, el2: %s|%s % (el1, el2)**

**list(map(func, [1, 2], [3, 4, 5]))   
*# >>> ['1|3', '2|4']***

**list1 = [7, 2, 3, 10, 12]**

**list2 = [-1, 1, -5, 4, 6]**

**print(\*map(lambda x, y: x\*y, list1, list2))**

**>>> -7, 2, -15, 40, 72**

Аналогічного (тільки при однаковій довжині списків) результату можна досягти за допомогою спискових виразів:

**[x\*y for x, y in zip(list1, list2)]**

**zip(sequence)** - функція, аналогічна map() в останньому варіанті, але може працювати з послідовностями різної довжини , повертає список кортежів:

**a = (1, 2, 3, 4)**

**b = (5, 6, 7, 8)**

**print(\*zip(a, b))**

**> >>(1, 5) (2, 6) (3, 7) (4, 8)**

**reduce( function, sequence)** повертає значення, отримане шляхом послідовного застосування бінарної функції function спочатку до перших двох елементів послідовності sequence, потім до результату і наступного елементу і т. д. Наприклад, обчислимо суму арифметичної послідовності:

**from functools import reduce**

**def add(x, y): return x+y**

**print ((reduce(add, range (1, 11)))))**

Обчислення суми всіх елементів списку за допомогою reduce:

**from functools import reduce**

**items = [1,2,3,4,5]**

**sum\_all = reduce(lambda x,y: x + y, items)**

**print (sum\_all)**

**15**

Обчислення найбільшого елемента у списку за допомогою reduce:

**from functools import reduce**

**items = [1, 24, 17, 14, 9, 32, 2]**

**all\_max = reduce(lambda a, b: a if (a > b) else b, items)**

**print (all\_max)**

**32**

**Стек та черги**

Список можна використовувати як стек - коли останній доданий елемент витягується першим (LIFO, last-in, first-out). Для отримання елемента з вершини стека є метод pop():

| **1**  **2**  **3**  **4**  **5**  **6** | **>>> stack = [1,2,3,4,5]**  **>>> stack.append (6)**  **>>> stack.append (7)**  **>>> stack.pop()**  **>>> stack**  **[1, 2, 3, 4, 5, 6]** |
| --- | --- |

Список можна використовувати як чергу - елементи витягуються у тому порядку, в якому вони додавалися (FIFO, first-in, first-out). Для отримання елемента використовується метод pop() з індексом 0:

| **1**  **2**  **3**  **4**  **5** | **>>> queue = ['rock','in','roll']**  **>>> queue.append ('alive')**  **>>> queue.pop(0)**  **>>> queue**  **['in', 'roll', 'alive']** |
| --- | --- |

**Кортежи (Tuple)**

Список також може бути незмінним (immutable), як і рядок, у такому випадку він зветься кортеж (tuple). Кортеж використовує менше пам'яті, ніж список. Кортеж замість квадратних дужок використовує круглі (хоча можна зовсім без дужок).

Створюємо порожній кортеж:

**>>> a = tuple() *# За допомогою вбудованої функції tuple()***

**>>> a**

**()**

**>>> a = () *# За допомогою літералу кортежу***

**>>> a**

**()**

**>>>**

Створюємо кортеж з одного елемента:

**>>> a = ('s')**

**>>> a**

**'s'**

Стоп. Вийшов рядок. Але як же так? Ми ж кортеж хотіли! Як же нам отримати кортеж?

**>>> a = ('s' , )**

**>>> a**

**('s',)**

Ура ! Запрацювало! Вся справа – у комі.

Ну і створити кортеж з об'єкта, що ітерується, можна за допомогою все тієї ж горезвісної функції tuple()

**>>> a = tuple('hello, world!')**

**>>> a**

**('h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '! ')**

Кортеж не допускає змін, до нього не можна додати новий елемент, хоча він може містити об'єкти, які можна змінити:

| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **>>> t = 1, [2,3]**  **>>> t**  **(1, [2, 3])**  **>>> t[1] = 2**  **TypeError: 'tuple' object does not support item assignment**  **>>> t[1].append(4)**  **>>> t**  **(1, [2, 3, 4])** |
| --- | --- |

Функція tuple() бере як аргумент рядок або список і перетворює його на кортеж:

| **1**  **2** | **>>> tuple('abc')**  **('a', 'b', 'c')** |
| --- | --- |

**Операції з кортежами**

Усі [операції над списками](https://pythonworld.ru/tipy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html) , які не змінюють список (додавання, множення на число, методи index() і count() та деякі інші операції). Можна також по-різному змінювати елементи місцями і так далі.

Наприклад , **swapping** :

**a, b = b, a**