**Масиви (cписки)**

**Масиви**

<https://www.e-olymp.com/uk/problems/8953>

<https://www.e-olymp.com/uk/problems/8954>

…

<https://www.e-olymp.com/uk/problems/8960> ++

…

<https://www.e-olymp.com/uk/problems/8970>

<https://www.e-olymp.com/uk/problems/8972>

…

<https://www.e-olymp.com/uk/problems/8976>

*Потреба запровадження табличних величин* випливає з необхідності запам'ятовувати й *однотипно* (за одним алгоритмом) опрацьовувати великий набір можливо *однотипних* даних. Наприклад, при знаходженні середнього балу кожного учня класу з інформатики за чверть потрібно знаходити суму великої кількості оцінок. Як зберігати всі ці оцінки? Зарезервувати для цього 40 чи більше змінних? Це дуже незручно.

У мові Python відповідні зручні структури даних називають *послідовностями*. Це *рядки* (символів), *списки, масиви, значення функції range* і ще деякі інші об'єкти.

Списку (у мові програмування) відповідає (у математиці) послідовність, члени якої занумеровано невід'ємними цілими числами, починаючи з 0, з *нефіксованими довжиною та типом елементів*.

**Оператори й функції для роботи зі списками** (s і t у позначеннях нижче)

* len(s) — довжина послідовності s;
* x in s — результат перевірка приналежності значення *x* послідовності *s* — True або False. У останніх версіях Python можна перевіряти приналежність підрядка рядку;
* x not in s — not (x in s) — результат заперечення x in s;
* s[j] — повертає значення *j*-го елемента списку *s* або (len(s)+j)-й при *j* < 0;
* s[j]= x — надання елементу з номером *j* списку *s* значення *x*;
* s[i:j:d] — зріз зі списку *s* від елементів *i*-го до j-го з кроком *d;*
* min(s) — найменший елемент *s;*
* max(s) — найбільший елемент *s;*
* s + t — результат «дописуванням» до списку *s* списку *t;*
* s\*n або n\*s — результат *n*-кратного повторення списку *s*. При *n* ≤ 0 — порожня послідовність.

Наприклад,

[1, 2, 3] + [4, 5] == [1, 2, 3, 4, 5]
[4, 5]\*3 = [4, 5, 4, 5, 4, 5]

**Методи списків**

* append(x) — додає елемент зі значенням *x* в кінець послідовності;
* clear — очищує список;
* copy — створює копію списку;
* count(x) — повертає кількість елементів зі значенням *x;*
* extend(s) — розширює список, додаючи до кінця поточного списку список *s;*
* index(x[,j0[,j1]]) — повертає найменший iндекс елемента зі значенням *x* [для номерів елементів від *j*0 [до *j*1]]. Породжує виключення *ValueError,* якщо елемента з таким значенням не знайдено;
* insert(j,x) — вставляє на місце з номером *j* елемент зі значенням *x;*
* pop(j) — повертає значення елемента з номером *j*, видаляючи його з послідовності;
* remove(x) — вилучає елемент з найменшим iндексом серед тих, що мають значення *x*. Породжує виключення *ValueError,* якщо елемента з таким значенням не знайдено;
* reverse() — зміняює порядок елементів на зворотний;
* sort([f]) — упорядковує елементи з можливістю використання власної функції порівняння *f;*

**Способи виведення елементів списку**

* **Вказівка print(a)** виводить елементи списку a у квадратних дужках через кому.
* **Цикл for** можна організувати двома способами:
	+ змінюючи індекс *j*, виводити елемент списку з індексом *j.*
	+ for i in range(len(A)):

 print(A[i], end = ' ')

(тут і далі запис end = ' ' означає вставлення пробілу);

* + змінюючи не індекс елемента списку, а його значення.
	+ for elem in A:

 print(elem, end = ' ')

В останньому прикладі бачимо істотну відмінність мови Pytnon від мов-попередників: цикл for надає зручний спосіб перебрати саме елементи *послідовності,* а не індекси елементів.

Розглянемо приклад створення списку d цілих чисел з цифри рядка s

s = 'ab12c59p7dq'

d = []

for с in s:

 if '1234567890'.find(с) != -1:

 d.append(int(с))

print(d)

з таким результатом виведення: [1, 2, 5, 9, 7].

**Способи створення й наповнення списків**

* **Використати багатократне повтороення списку** для заповнення списку одними й тими самими значеннями. Наприклад,
* n = 999

a = [0] \* n

* **Перелічити елементи списку у квадратних дужках.** Наприклад,
* Primes = [2, 3, 5, 7, 11, 13]

Rainbow = ['червоний','оранжевий','жовтий','зелений','блакитний','синій','фіолетовий']

* **Заповнити список згідно з формулою**

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

де *послідовність* містить усі значення, яких набуває *змінна*, від якої залежить *вираз*. Наприклад, список квадратів усіх цифр [0, 1, 4, 9, 16, 25, 36, 49, 64, 81] можна створити таким чином:

a = [j\*\*2 for j in range(0,10)]

print(a)

* **Cтворити порожній список і в кінець списку додавати елементи (по одному) за допомогою методу append,** значення яких зчитувати, наприклад, з клавіатури.
* a = [] # створено порожній список
* n = int(input()) # зчитино кількість елементів списку
* for i in range(n):

 a.append(int(input()))

Змінну *n* використано лише для зручності сприйняття — див. змінений приклад.

A = []

for i in range(int(input()):

 A.append(int(input())

Можна по-іншому організувати процес прочитування списків, спочатку створивши список з потрібного числа елементів.

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

for i in range(len(A)):

 A[i] = int(input())

* **Зчитати з файлу**
* f=open('input.txt','r')
* a=[]
* for i in range(n):
* c=f.readline(n)[0:-1]
* a.append(c)

print(a)

* **Заповнити випадковими числами** з використанням функциї randrange з модуля random:
* from random import randrange
* n = 10

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

**Обчислення суми елементів масиву, що задовольняють певним умовам**
(приклад подано для знаходження суми парних елементів лінійного масиву a, заповненого випадковими числами)

import random

n=9

amax=99

a=[]

for j in range(n):

 a.append(random.randint(1,amax))

print(a)

print('Сума парних чисел з поданих дорівнює')

s=0

for j in range(n):

 if a[j]%2==0:

 s+=a[j]

print(s)

**Визначення найменшого елемента масиву**
Ідея пошуку така: змінній min — найменшому значенню з проглянутих — спочатку надають значення елемента масиву з найменшим номером. Послідовно переглядаючи значення наступних елементів масиву при виявленні значення, меншого від min, надаємо змінній min цього значення.

import random

n=9

amax=99

a=[]

for j in range(n):

 a.append(random.randint(1,amax))

print(a)

print('Найменше з поданих чисел дорівнює')

Min=a[0]

for j in range(n):

 if a[j]<Min:

 Min=a[j]

print(Min)

**Впорядкування елементів масиву вибором найменшого елемента**
(опис подано для лінійного масиву з діапазоном індексу 1..n)

Змінюючи *j* від 1 — найменшого значення індексу — до (*n* – 1) — найбільшого значення індексу, зменшеного на 1, — робимо таке:

* визначаємо номер *l* найменшого елемента для індексів у межах від *j* до *n* включно;
* міняємо місцями значення елементів з індексами *j* та *l*. У результаті такої заміни на місце з номером (індексом) *j* стане те значення, що потрібно.

Подамо для наочності ілюстрацію такого впорядкування 10-елементного масиву, запози­чену зі [сторінки](http://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%B8%D0%B1%D0%BE%D1%80%D0%BE%D0%BC) Вікіпедії.



Тут червоним тлом виділено те значення, яке серед елементів з номером від *j* до *n* вважають найменшим у поточний момент виконання алгоритму, блакитним — значення поточного елемента масиву, яке порівнюють з тим, що наразі вважають найменшим, жовтим — ті значення, які вже стоять на потрібному місці.

Код програми мовою Python має такий вигляд.

import random

n=9

amax=99

a=[]

for j in range(n):

 a.append(random.randint(1,amax))

print(a)

for j in range(n-1):

 for k in range(j+1,n):

 if a[j]>a[k]:

 a[j],a[k]=a[k],a[j]

print('Результат впорядкування цього масиву такий')

print(a)

**Метод швидкого сортування**
Основу цього алгоритму розробив у 1962 році британський учений Чарлз Ентоні Річард Гоар (Charles Antony Richard Hoare).

*Рекурсивний* алгоритм для впорядкування за неспаданням масиву зі значеннями індексів від *l* до *r* включно має такий вигляд:

1. Вибирати значення елемента масиву.
2. Значення елементів масиву, що не менші за вибране, розташувати на місцях з номерами (індексами) більшими від індексу обраного значення, а ті, що не перевищують — на місцях з номерами меншими від індексу обраного значення. У результаті вибране значення матиме той номер *j* у масиві, що потрібно. Тому надалі відповідний елемент масиву буде незмінним.
3. Якщо *l* < (*j* – 1), виконати алгоритм для індексів від *l* до (*j* – 1) включно.
4. Якщо (*j* + 1) < *r*, виконати алгоритм для індексів від (*j* + 1) до *r* включно.

**Рекурсивність алгоритму** *означає виклик цього алгоритму в ньому самому* — див. кроки 3–4. На малюнку нижче подано *частину* схеми впорядкування.

Тут кольоровими кругами позначено вибрані значення елементів масиву. Після переставлянь навколо них ці значення надалі залишаються нерухомими у масиві. Спочатку буде впорядковано елементи, менші від значення, позначеного зеленим кольором, потім — менші від значення, позначеного червоним кольором, ще пізніше — менші від значення, позначеного синім кольором.

**Примітка.** Поданий вище запис стане алгоритмом лише після того, як буде деталізовано вибір значення. У багатьох описах алгоритму можна зустріти вибір елемента з найменшим чи з найбільшим номером. Розглянемо спробу впорядкувати вже впорядкований масив при такому виборі. Глибина занурення — кількість послідовних викликів рекурсивного алгоритму без виходу з нього — буде майже збігатися з довжиною масиву. Останнє може призвести до переповнення стеку частини оперативної пам'яті, куди програма заносить дані про виклик програмі та функцій, їхні аргументи, точку повернення тощо. А переповнення стеку означає аварійне завершення роботи. Який би не був детерміністичний (без випадковостей) вибір значення (наприклад, із середини масиву), завжди можна вказати вхідні дані, для яких глибина занурення для цих вхідних даних лише на 1 відрізнятиметься від довжини масиву. Тому *вибір потрібно робити з використанням генератора випадкових чисел.* І в цьому випадку можливе переповнення стеку, але ймовірність цієї події настільки мала, що аварійне припинення роботи зазвичай не відбувається.

Проаналізуйте на відповідність опису ілюстрацію для вибору правого краю (елемента з найбільшим номером), запозичену із сайту [http://uk.wikipedia.org/wiki](http://uk.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B8%D0%B4%D0%BA%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)



і код програми.

import random

n=9

amax=99

a=[]

for j in range(n):

 a.append(random.randint(1,amax))

print(a)

def qsort(L):

 if L: return qsort([x for x in L if x<L[0]]) + [x for x in L if x==L[0]] + qsort([x for x in L if x>L[0]])

 return []

print('Результат впорядкування цього масиву такий')

print(qsort(a))

При прямуванні *n* — довжини лінійного масиву — до нескінченності кількість виконуваних операцій при впорядкуванні (майже) пропорційна:

* *n*2 — для методів впорядкування вибором найменшого елемента або бульбашки і шейкерного, не описаних у цій розробці;
* *n* log2*n* — для методів впорядкування Хоара або пірамідального і злиттям, не описаних у цій розробці;
* *n* + *m* — для методу обліку з *m* — кількістю елементів діапазону зміни *значень* елементів масиву.

Іншими словами про це кажуть так: часові складності алгоритмів дорівнюють відповідно *O*(*n*2), *O*(*n* log2*n*) і *O*(*n* + *m*). У більшості випадків доречно використати саме швидкий метод Хоара, хоча бувають випадки, коли для розв'язування поставленої задачі (не лише упорядкування) потрібно використати, наприклад, метод злиття, або використати метод обліку, щоб задовольнити обмеження на час виконання програми. Обидва випадки зустрічалися на ІІІ (міському) етапі Всеукраїнської учнівської олімпіади з інформатики у місті Києві.

**Метод обліку** передбачає використання окремого масиву з нульовими початковими значеннями, який після проглядання масиву, що підлягає впорядкуванню, містить інформацію про те, скільки разів те чи інше значення зустрілося серед значень елементів масиву.

import random

n=10

amax=20

a=[]

for j in range(n): a.append(random.randint(1,amax))

print(a)

b=[0]\*(amax+1)

for j in a: b[j]+=1

c=[]

for j in range(amax+1):

 if b[j]>0:

 for k in range(b[j]):

 c.append(j)

print(c)

**Вбудований алгоритм упорядкування** з довгими списками працює, як упорядкування злиттям, а на коротких фрагментах — як упорядкування вставками. Упорядковує будь-які об'єкти, які можна порівнювати: числа, рядки (їх порівнюють лексикографічно), списки, кортежі. Можна оголосити (описати) власний клас об'єктів, визначити для них функцію порівняння й упорядковувати їх за допомогою цього методу. Є два способи застосування цього алгоритму. Перший спосіб — це *метод* sort, застосований до списку. Наприклад:

import random

n=9

amax=99

a=[]

for j in range(n):

 a.append(random.randint(1,amax))

print(a)

a.sort()

print(a)

У цьому випадку елементи списку буде переставлено. Метод sort не повертає значення, тому результат виклику методу sort не використовують в арифметичних виразах або при виведенні результату. Інший спосіб — це *функція* sorted. Вона, навпаки, не модифікує переданий їй список, а повертає новий список. Функцію sorted можна викорис­товувати таким чином:

import random

n=9

amax=99

a=[]

for j in range(n):

 a.append(random.randint(1,amax))

print(a)

b=sorted(a)

print(b)

Функцію sorted можна використати при введенні-виведенні. Наступна програма упорядковує зчитаний з одного рядка введення список чисел, розділених у рядку пробілами, і виводить результат на екран.

print ( " " .join (map (str, sorted (map (int, input (). split ())))))

І методу sort, і функції sorted можна передати значення параметра reverse. Якщо це значення дорівнює True, то упорядкування буде проведено у зворотному порядку, тобто за незростанням.

Якщо елементи списку самі є списками, то сортування відбуватиметься за першими елементам вкладених списків:

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

>>>print(sorted(a))

[[1, 3], [2, 1], [3, 2]]

При потребі упорядкувати за іншим елементом використовують необов'язковий аргумент key, якому надають значення — назву функції, що повертає критерій упорядкування. Все це можна побачити у наступному прикладі програми опрацювання бази даних, кожний елемент якої містить дані про юного спортсмена: ім'я, вік (у роках), зріст (у см) і вагу (у кілограмах).

a = [['Петро',10,130,35], ['Василина',11,135,39],['Христина',9,140,33],['Тарас',10,128,30]]

n = input('Упорядкувати за іменем (1), віком (2), зростом (3) чи вагою (4): ')

n = int(n)-1

def sort\_col(i):

 return i[n]

a.sort(key=sort\_col)

for i in a:

 print(i[0],' ', i[1],' ', i[2],' ',i[3])

Якщо критерієм упорядкування елемент вкладеного списку (як у попередньому випадку), то можна використати функцію lambda

a = [['Петро',10,130,35], ['Василина',11,135,39],['Христина',9,140,33],['Тарас',10,128,30]]

n = input('Упорядкувати за іменем (1), віком (2), зростом (3) чи вагою (4): ')

n = int(n)-1

a.sort(key=lambda i: i[n])

for i in a:

 print(i[0],' ', i[1],' ', i[2],' ',i[3])

**(Бінарний) пошук заданого елемента**
Розглянемо задачу пошуку номера елемента упорядкованої за *зростанням* послідовності {*an*}, який (елемент) має значення *b*. Зазвичай для цього викорис­товують двійковий (бінарний) пошук. Він полягає у послідовному поділі (приблизно) навпіл діапазону номерів, у якому проводять пошук. На початку виконання цього алгоритму:

* *l* — менша межа діапазону номерів, на якому шукають потрібний номер, дорівнює меншій межі діапазону номерів масиву;
* *r* — більша межа діапазону номерів, на якому шукають потрібний номер, дорівнює більшій межі діапазону номерів масиву.

**Алгоритм бінарного пошуку** має такий вигляд:

1. Поки справджується нерівність *l* ≤ *r*, робити таке:
	* надати значення змінній *l* = [(*j* + *k*)/2] — знайти індекс *l*, що поділяє досліджуваний діапазон навпіл;
	* якщо *al = b*, завершити пошук зі знайденим номером *l*;
	* якщо *al < b*, надати значення змінній *l* = (*x* + 1), тобто вибрати для подальшого пошуку діапазон, розташований правіше від *l*;
	* якщо *al > b*, надати значення змінній *r* = (*x* – 1), тобто вибрати для подальшого пошуку діапазон, розташований лівіше від *l*;
2. Завершити пошук, вважаючи його неуспішним.

import random

import math

a=[]

n=10

for i in range(n):

 a.append(random.randint(1,100))

print(a)

k=a[3]

l=0

r=n-1

while True:

 x=math.ceil((l+r)/2)

 if k==a[x]:

 res=x

 break

 elif k<a[x]:

 r=x

 else:

 l=x

print(res)

Якщо розмір і тип елементів послідовності не знінні протягом виконання програми, то використовують модуль роботи з масивами [array](http://www.kievoit.ippo.kubg.edu.ua/kievoit/2016/43_Python/index.html#array).

|  |  |
| --- | --- |
| **aa=dict()****for i in range(10):** **for j in range(10):** **aa[i,j]=i\*j****for i in range(10):** **for j in range(10):** **print('%4d' % aa[i,j], end='')** **print()** |  |

<https://www.e-olymp.com/uk/problems/4730>

Фібоначчі

Числа Фібоначчі - це послідовність чисел F(**n**), яка задається формулою:

F(**0**) = **1**, F(**1**) = **1**, F(**n**) = F(**n** - **1**) + F(**n** - **2**)

За заданим числом **n** вивести **n**-те число Фібоначчі.

**Вхідні дані**

Невід'ємне число **n** (**n** ≤ **45**) - номер числа Фібоначчі, яке потрібно вивести.

**Вихідні дані**

Вивести **n**-те число Фібоначчі.

Ліміт часу **1** секунда

Ліміт використання пам'яті **128** MiB

**Вхідні дані #1**

4

**Вихідні дані #1**

5

|  |  |
| --- | --- |
| n=int(input())a=[]a.append(1)a.append(1)for i in range(2,n+1): a.append(a[i-1]+a[i-2])print(a[n])  | n=int(input())a=dict()a[1]=1a[2]=1for i in range(3,n+2): a[i]=a[i-1]+a[i-2]print(a[n+1])  |

<https://www.e-olymp.com/uk/problems/8960>

Крім найменших і найбільших

З консолі вводиться масив з **N** цілих чисел. Програма повинна знайти суму елементів масиву, не враховуючи всіх його найменших і найбільших елементів.

**Вхідні дані**

В першому рядку записане натуральне число **N**. У наступному рядку записані **N** цілих чисел. Всі числа за модулем не перевищують **100**.

**Вихідні дані**

Відповідь до задачі.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

**Вхідні дані #1**

7

6 2 7 1 7 1 2

**Вихідні дані #1**

10

|  |  |
| --- | --- |
| Python |  |
| n=int(input())a=list(map(int,input().split()))mmax=a[0]mmin=a[0]for i in range(n): if mmax>a[i]: mmax=a[i] if mmin<a[i]: mmin=a[i]s=0for i in range(n): if a[i]!=mmin and a[i]!=mmax: s=s+a[i] print(s) |  |

<https://www.e-olymp.com/uk/problems/4746>

Кафе "Хоботанія"

У місті Є. відкрилось нове кафе "Хоботанія", розраховане на слоників. Усі клієнти приходять у кафе в момент часу **0**, а господар вибирає у якому порядку їх обслуговувати. При цьому кожну секунду обслуговується один слоник (перший обслуговується у момент часу **0**).

Господару відомо, що якщо слоника **i** буде обслужено у момент часу **t**, то він заплатить tipsi - **t** чайових. Якщо число tipsi - **t** від'ємне, то він нічого не платить.

Допоможіть господару знайти такий порядок обслуговування слоників, який принесе йому максимальний прибуток.

**Вхідні дані**

Перший рядок містить кількість слоників **n** (**0** ≤ **n** ≤ **100**), що прийшли у кафе. Наступний рядок містить **n** чисел tips1, tips2, ..., tipsn (**0** ≤ tipsi ≤ 105).

**Вихідні дані**

Виведіть максимальний прибуток володаря кафе.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

**Вхідні дані #1**

3

3 2 3

**Вихідні дані #1**

5

|  |
| --- |
| x = 0n = int(input())if n>0: arr = sorted(list(map(int, input().split())), key=int, reverse=True) for i in range(n): if arr[i] - i > 0: x += arr[i] - iprint(x) |