**Списки**

**Списки**

**Cписки** (в большинстве же языков программирования используется другой термин — «массив») — это такая структура данных, которая позволяет хранить в себе несколько однотипных значений, таких как числа или строки. Например, если мы хотим проверить число на простоту, а потом построить его разложение на простые множители, нам удобно создать список простых чисел. Тогда мы сможем проверять делимость числа на все ранее построенные элементы списка.

Список в Python представляет собой последовательность элементов, пронумерованных от 00, как символы в строке. Тогда список, состоящий из nn элементов, будет заканчиваться элементом с индексом n−1n−1.

**Пример:**

Список можно задать перечислением элементов списка в квадратных скобках, например, список из первых 66 простых чисел можно задать так:

p = [2, 3, 5, 7, 11, 13]

В списке p— 66 элементов, а именно, p[0] == 2, p[1] == 3, p[2] == 5, p[3] == 7, p[4] == 11, p[5] == 13.

Также как и символы строки, элементы списка в Python можно индексировать отрицательными числами с конца, например, p[-1] == 13. Длину списка, то есть количество элементов в нём, можно узнать при помощи функции len(), например, len(p) == 6.

Если мы попробуем обратиться к элементу списка по некорректному индексу, например, обратимся к элементу p[6], то произойдёт ошибка **IndexError: list index out of range**.

**Пример:**

Пусть теперь мы хотим перебрать все элементы списка, при этом выводя их на экран, а потом возводя в квадрат. Организуем цикл следующим образом:

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

**print**(p[i])

p[i] = p[i] \*\* 2

Таким образом, при использовании списков можно присваивать элементам новые значения, в отличие от работы со строками.

**Пример:**

Пусть нам необходимо хранить базу данных учащихся некоторой школы, содержащую данные об учащихся и их оценках для ведения электронного дневника. Заведем списки для хранения имен учащихся names и для хранения их оценок scores, причём элементы с одинаковыми индексами будут соответствовать один и тем же ученикам. Выведем данные, пронумеровав записи, с помощью следующей программы:

names = ['Иванов', 'Петров', 'Королев']

scores = [5, 4, 5]

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

**print**(i+1, '.', sep = '', end = ' ')

**print**(names[i], scores[i])

На выходе получим:

1. Иванов 5

2. Петров 4

3. Королев 5

**Операции со списками**

Для списков целиком определены следующие операции: конкатенация списков (добавление одного списка в конец другого) и повторение списков (умножение списка на число). Например:

a = [1, 2, 3]

b = [4, 5]

c = a + b

d = a \* 10

В результате приведенных выше операций список c будет равен [1, 2, 3, 4, 5], а список d будет состоять из 3030 элементов, из элементов 1, 2, 3, повторенных 1010 раз.

**Создание списков**

Операция повторения списка позволяет конструировать большие списки. Например, списки с 10001000 элементов, заполненных нулями.

a = [0] \* 1000

Рассмотрим другой способ создания списков — при помощи метода append, позволяющего помещать элементы в конец списка. Прежде всего можно создать пустой список (не содержащий элементов, имеющий длину 00), в конец списка можно последовательно добавлять элементы.

a = []

**for** i **in** range(1000):

a.append(0)

Ноль можно заменить любым другим выражением.

a = []

**for** i **in** range(1000):

a.append(i)

В полученном списке будет 10001000 элементов — от 0 до 999.

**Методы split и join**

**Метод split**

Элементы списка, разделённые пробелами, могут при вводе находиться в одной строке. В этом случае строку можно считать функцией input(). После этого можно использовать метод строки split, возвращающий список строк, разрезав исходную строку на части по пробелам.

Пример:

a = input().split()

Если при запуске этой программы ввести строку '2 3 5 7 11', то список a будет равен ['2', '3', '5', '7', '11']. Обратите внимание, что список будет состоять из строк, а не из чисел! Если хочется получить список именно из чисел, то можно элементы списка по одному преобразовать в числа:

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

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

В результате получим список, состоящий из чисел: [2, 3, 5, 7, 11]

Используя функции языка map() и list() то же самое можно сделать в одну строку:

a = list(map(int, input().split()))

Здесь манипулятор map() применяет функцию int() сразу ко всем элементам, полученным после разделения строки, и организует из них список list. Если нужно считать список действительных чисел, то в данной записи необходимо заменить int() на float().

Подобным способом можно считать и несколько отдельных переменных из одной строки. В этом случае распаковка по переменным считанных данных произойдет автоматически:

n, k = map(int, input().split())

**Метод join**

Научимся выводить элементы списка в одной строке через пробел при помощи однострочной команды. Для этого используется метод join. У этого метода один параметр — список строк. В результате получается строка, которая является соединением элементов списка (которые были переданы в качестве параметра). При этом между элементами списка вставляется разделитель, равный той строке, к которой применяется метод. Например, программа

a = ["red", "green", "blue"]

**print**(" ".join(a))

**print**("\*\*\*".join(a))

**print**("".join(a))

выведет строки "red green blue", "red\*\*\*green\*\*\*blue" и "redgreenblue".

Следующая программа выведет считанные слова в столбец по одному в каждой строке. Здесь символ **\n** обзначает переход на новую строчку:

**print**("**\n**".join(input().split()))

Если же список состоит из чисел, то придется использовать функцию map(). То есть вывести элементы списка чисел, разделяя их пробелами, можно, предварительно преобразовав их в строки следующим образом:

**print**(" ".join(map(str, a)))

**Линейный поиск**

**Задача**

Необходимо найти индекс первого вхождения некоторого элемента key в заданный массив, а если такого элемента в массиве нет, то напечатать сообщение 'нет'.

**1-й способ решения**

key = int(input())

a = list(map(int, input().split()))

idx = -1

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

**if** a[i] == key:

idx = i

**print**(idx)

**break**

**if** idx == -1:

**print**("нет")

**2-й способ решения**

key = int(input())

a = list(map(int, input().split()))

found = False

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

**if** a[i] == key:

**print**(i)

found = True

**break**

**if** **not** found:

**print**("нет")

**3-й способ решения**

key = int(input())

a = list(map(int, input().split()))

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

**if** a[i] == key:

**print**(i)

**break**

**else**:

**print**("нет")

**4-й способ решения**

key = int(input())

a = list(map(int, input().split()))

i = 0

**while** i < len(a) **and** a[i] != key:

i += 1

**if** i < len(a):

**print**(i)

**else**:

**print**("нет")

**Генераторы**

**Генераторы**

Для создания списка, заполненного одинаковыми элементами, можно использовать оператор повторения списка, например:

а = [0] \* n

Для создания списков, заполненных по более сложным формулам, можно использовать **генераторы** — выражения, позволяющие заполнить список некоторой формулой. Общий вид генератора следующий:

[выражение **for** переменная **in** список]

Здесь переменная — идентификатор некоторой переменной, список — список значений, который принимает данная переменная (как правило, полученный при помощи функции range()), выражение — некоторое выражение, которым будут заполнены элементы списка, как правило, зависящее от использованной в генераторе переменной.

Вот несколько примеров использования генераторов.

Создать список, состоящий из nn нулей можно и при помощи генератора:

a = [0 **for** i **in** range(n)]

Создать список, заполненный квадратами целых чисел, можно так:

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

Если нужно заполнить список квадратами чисел от 11 до nn, то можно выражение следующим образом:

a = [(i + 1) \*\* 2 **for** i **in** range(n)]

Другим способом может быть изменение параметров функции range():

a = [i \*\* 2 **for** i **in** range(1, n + 1)]

Таким образом можно генерировать достаточно сложные выражения и заполнять элементы списка по определенным правилам.

**Модуль random**

Иногда возникает необходимость использования случайных чисел. Для этого можно использовать, например, функцию randint из модуля random. В начале вашей программы следует подключить этот модуль с помощью команды:

**import** random

**Пример:**

Смоделируем nn бросков кубика, то есть сгенерируем список, заполненный случайными числами от 11 до 66 включительно:

a = [random.randint(1, 6) **for** i **in** range(n)]

**Cчитывание данных**

Рассмотрим еще одно применение списочных выражений — считывание данных. Пусть список состоит из строк, которые состоят из чисел. Сначала нужно ввести число элементов списка (это значение будет использовано в качестве аргумента функции range()), потом — заданное количество строк. Программа для их считывания будет выглядеть следующим образом:

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

Если список чисел дан в одной строке через пробел, то можно использовать следующую конструкцию: a = list(map(int, input().split())). Используя генераторы, можно сделать то же самое с использованием следующей конструкции:

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

В данном случае индексная переменная elem принимает значения, равные элементам входной строки, разбитой пробелами. После преобразования этих элементов функцией int получим список из целых чисел.

**Срезы**

### Срезы

Для списков, как и для строк, определена операция взятия среза. А именно:

* a[i : j] — срез из j−ij−i элементов: a[i], a[i + 1],..., a[j - 1].
* a[i : j : k] — срез с шагом kk: a[i], a[i + k], a[i + 2 \* k],....

Если значение k<0k<0, то элементы будут идти в противоположном порядке.

Любое из чисел i или j может отсутствовать, что означает «начало списка» или «конец списка».

Списки, в отличие от строк, являются изменяемыми объектами: можно отдельному элементу списка присвоить новое значение. Можно поменять и целиком срез. Рассмотрим примеры:

>>>a = [1, 2, 3, 4, 5, 6]

>>>a[2 : 4] = [7, 8, 9, 10]

>>>**print**(a)

[1, 2, 7, 8, 9, 10, 5, 6]

>>>a[-2 : ] = []

>>>**print**(a)

[1, 2, 7, 8, 9, 10]

>>>a[1 : : 2] = [11, 12, 13]

>>>**print**(a)

[1, 11, 7, 12, 9, 13]

Если срезу с шагом kk, отличным от 11, присвоить новое значение, то количество элементов в старом и новом срезе обязательно должно совпадать, в противном случае произойдёт ошибка **ValueError**.

Обратите внимание, что операция обращения к элементу по индексу отличается от взятия среза. a[i] — это элемент списка, а не срез!

>>>a = [1, 11, 7, 12, 9, 13]

>>>a[2] = [14, 15]

>>>**print**(a)

[1, 11, [14, 15], 12, 9, 13]

**Операции со спискам**

Со списками можно легко делать много разных операций. Рассмотрим основные из них:

|  |  |
| --- | --- |
| min(A) | Наименьший элемент списка. Элементы списка могут быть числами или строками, для строк сравнение элементов проводится в лексикографическом порядке. |
| max(A) | Наибольший элемент списка. |
| sum(A) | Сумма элементов списка. Элементы обязательно должны быть числами. |
| x **in** A | Проверить, содержится ли элемент в списке. Возвращает True или False . |
| x **not in** A | То же самое, что **not**(x **in** A). |
| A.index(x) | Индекс первого вхождения элемента x в список, при его отсутствии генерирует исключение ValueError. |
| A.count(x) | Количество вхождений элемента x в список. |
| A.append(x) | Добавить в конец списка A элемент x. |
| A.insert(i, x) | Вставить в список A элемент x на позицию с индексом i. Элементы списка A, которые до вставки имели индексы i и больше, сдвигаются вправо. |
| A.extend(B) | Добавить в конец списка A содержимое списка B. |
| A.pop() | Удалить из списка последний элемент, возвращается значение удалённого элемента. |
| A.pop(i) | Удалить из списка элемент с индексом i, возвращается значение удаленного элемента. Все элементы, стоящие правее удаленного, сдвигаются влево. |
| A.remove(x) | Удаляет первое вхождение элемента x. Если элемента x нет в списке,то получаем ошибку во время выполнения. |