

Списки

Большинство программ работает не с отдельными переменными, а с набором переменных. Например, программа может обрабатывать информацию об учащихся класса, считывая список учащихся с клавиатуры, при этом изменение количества учащихся в классе не должно требовать модификации исходного кода программы.

Раньше мы сталкивались с задачей обработки элементов последовательности, например, вычисляя наибольший элемент последовательности. Но при этом мы не сохраняли всю последовательность в памяти компьютера, однако, во многих задачах нужно именно сохранять всю последовательность, например, если бы нам требовалось вывести все элементы последовательности в возрастающем порядке (“отсортировать последовательность”).

Для хранения таких данных можно использовать структуру данных, называемую в Питоне **список** (в большинстве же языков программирования используется другой термин – “массив”).

Массив — это составной объект, образованный из элементов (*компонент*) одного и того же типа. Такой тип данных применяется в программировании для обозначения объектов, аналогичных числовым последовательностям в математике, где сразу целая группа чисел или других объектов обозначается одним именем (чаще всего буквой), а для обращения к каждому отдельному числу данной последовательности используются их *индексы* (номера элементов). В математике это выглядит, например, так: $a_0, a_1, a_2, a_3, \dots, a_n$.

Список в Python представляет собой последовательность элементов, пронумерованных от 0, как символы в строке. Тогда, список, состоящий из n элементов, будет заканчиваться элементом с индексом $n - 1$. Список можно задать перечислением элементов списка в квадратных скобках, например, список можно задать так:

```
Primes = [2, 3, 5, 7, 11, 13]
Rainbow = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']
```

В списке Primes — 6 элементов, а именно,

```
Primes[0] == 2, Primes[1] == 3, Primes[2] == 5, Primes[3] == 7,
Primes[4] == 11, Primes[5] == 13. Список Rainbow состоит из 7 элементов,
каждый из которых является строкой.
```

Также как и символы строки, элементы списка в Python можно индексировать отрицательными числами с конца, например, `Primes[-1] == 13, Primes[-6] == 2`.

Длину списка, то есть количество элементов в нем, можно узнать при помощи функции `len`, например, `len(A) == 6`.

Рассмотрим несколько способов создания и считывания списков. Прежде всего можно создать пустой список (не содержащий элементов, длины 0), в конец списка можно добавлять элементы при помощи метода `append`. Например, если программа получает на вход количество элементов в списке n , а потом n элементов списка по одному в отдельной строке, то организовать считывание списка можно так:

```
A = []
N = int(input())
for i in range(N):
    A.append(int(input()))
```

В этом примере создается пустой список, далее считывается количество элементов в списке, затем по одному считываются элементы списка и добавляются в его конец.

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 = B * 3
```

В результате приведенных выше операций список C будет равен [1, 2, 3, 4, 5], а список D будет равен [4, 5, 4, 5, 4, 5]. Это позволяет по-другому организовать процесс считывания списков: сначала считать размер списка и создать список из нужного числа элементов, затем организовать цикл по переменной i начиная с числа 0 и внутри цикла считывается i-й элемент списка:

```
A = [0] * int(input())
for i in range(len(A)):
    A[i] = int(input())
```

Вывести элементы списка A можно одной инструкцией print(A), при этом будут выведены квадратные скобки вокруг элементов списка и запятые между элементами списка. Такой вывод неудобен, чаще требуется просто вывести все элементы списка в одну строку или по одному элементу в строке. Приведем два примера, также отличающиеся организацией цикла:

```
for i in range(len(A)):
    print(A[i])
```

Здесь в цикле меняется индекс элемента i, затем выводится элемент списка с индексом i.

```
for elem in A:
    print(elem, end = ' ')
```

В этом примере элементы списка выводятся в одну строку, разделенные пробелом, при этом в цикле меняется не индекс элемента списка, а само значение переменной (например, в цикле for elem in ['red', 'green', 'blue'] переменная elem будет последовательно принимать значения 'red', 'green', 'blue').

Для вывода всех элементов списка в одной строке без квадратных скобок и запятых в современных версиях компилятора существует и совсем простой способ:

```
print(*A)
```

Методы split и join

Элементы списка могут при вводе находиться и в одной строке, разделенные, например, пробелами. 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 `1 2 3`, то список `A` будет равен `['1', '2', '3']`. Обратите внимание, что список будет состоять из строк, а не из чисел!!! Если хочется получить список именно из чисел, то можно затем элементы списка по одному преобразовать в числа:

```
for i in range(len(A)):
    A[i] = int(A[i])
```

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, организуя из них список (`list`). Если нужно считать список действительных чисел, то в данной записи необходимо заменить тип `int` на тип `float`.

У метода `split` есть необязательный параметр, который определяет, какая строка будет использоваться в качестве разделителя между элементами списка. Например, метод `split('.')` вернет список, полученный разрезанием исходной строки по символам `'.'`.

Используя “обратные” методы можно вывести список при помощи однострочной команды. Для этого используется метод строки `join`. У этого метода один параметр: список строк. В результате получается строка, полученная соединением элементов списка (которые переданы в качестве параметра) в одну строку, 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'`, `redgreenblue` и `red***green***blue`.

Если же список состоит из чисел, то придется использовать еще и функцию `map`. То есть вывести элементы списка чисел, разделяя их пробелами, можно, предварительно преобразовав их в строки так:

```
print(' '.join(map(str, A)))
```

Ввод. Обобщение

Таким образом, если нам необходимо считать числа из одной строки, то можно использовать такой метод:

```
A = list(map(int, input().split()))
```

А если нам надо создать список из n значений (где n задаётся в первой строке), записанных в разных строках, тогда можно воспользоваться таким методом:

```
A = []
for i in range(int(input())):
    A.append(int(input()))
```

Генераторы списков

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

```
A = [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 ]
```

где *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 n 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 1 до n , то можно изменить параметры функции `range` на `range(1, n + 1)`:

```
A = [ i ** 2 for i in range(1, n + 1) ]
```

Вот так можно получить список, заполненный случайными числами от 1 до 9 включительно (используя функцию `randint` из модуля `random`):

```
A = [ randint(1, 9) for i in range(n) ]
```

А для того чтобы использовать этот модуль, в начале вашей программы следует его подключить с помощью команды:

```
from random import *
```

В последнем примере список будет состоять из строк, считанных со стандартного ввода: сначала нужно ввести число элементов списка (это значение будет использовано в качестве аргумента функции `range`), потом — заданное количество строк:

```
A = [ input() for i in range(int(input())) ]
```