Списки в Python

Информатика. Язык программирования Python в курсе информатики. Все классы.

Большинство программ работает не с отдельными переменными, а с набором переменных. Например, программа может обрабатывать информацию об учащихся класса, считывая список учащихся с клавиатуры или из файла, 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”). Список представляет собой последовательность элементов, пронумерованных от 0, как символы в строке. 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 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 = []
for i in range(int(input())):
   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 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’.

Операции со списками в Python

Информатика. Язык программирования Python в курсе информатики. Все классы.

|  |  |
| --- | --- |
| 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) |
| min(A) | Наименьший элемент списка |
| max(A) | Наибольший элемент списка |
| 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.sort() | Сортировка списка (меняет сам список, ничего не возвращает) |
| sum(A) | Возвращает сумму элементов в списке |
| A.append(x) | Добавить элемент x в конец списка |
| A.extend(L) | Добавить все элементы списка L в конец списка A |

Срезы списков в Python

Информатика. Язык программирования Python в курсе информатики. Все классы.

Со списками, так же как и со строками, можно делать срезы. А именно:

A[i:j] срез из j-i элементов A[i], A[i+1], …, A[j-1].

A[i:j:-1] срез из i-j элементов A[i], A[i-1], …, A[j+1] (то есть меняется порядок элементов).

A[i:j:k] срез с шагом k: A[i], A[i+k], A[i+2\*k],… . Если значение k<0, то элементы идут в противоположном порядке.

Каждое из чисел i или j может отсутствовать, что означает “начало строки” или “конец строки”

Списки, в отличии от строк, являются изменяемыми объектами: можно отдельному элементу списка присвоить новое значение. Но можно менять и целиком срезы. Например:

A = [1, 2, 3, 4, 5]
A[2:4] = [7, 8, 9]

Получится список, у которого вместо двух элементов среза A[2:4] вставлен новый список уже из трех элементов. Теперь список стал равен [1, 2, 3, 7, 8, 9, 5].

A = [1, 2, 3, 4, 5, 6, 7]
A[::-2] = [10, 20, 30, 40]

Получится список [40, 2, 30, 4, 20, 6, 10]. Здесь A[::-2] — это список из элементов A[-1], A[-3], A[-5], A[-7], которым присваиваются значения 10, 20, 30, 40 соответственно.

Если не непрерывному срезу (то есть срезу с шагом k, отличному от 1), присвоить новое значение, то количество элементов в старом и новом срезе обязательно должно совпадать, в противном случае произойдет ошибка ValueError.

Для непрерывных срезов количество элементов может не совпадать, что привет к добавлению элементов в середину списка или удалению элементов из середины. Это может быть затратной по времени операцией, если для вставки или удаления необходимо переместить в памяти значительное количество элементов.

Обратите внимание, A[i] — это операция обращения к одному элементу списка, а не к срезу!

Генераторы списков в Python

Информатика. Язык программирования Python в курсе информатики. Все классы.

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 (используя функцию randint из модуля random):

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

А в этом 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()))]



Обращение массива

Информатика. Язык программирования Python в курсе информатики. Все классы.

Для того, чтобы переставить элементы массива A длины N в обратном порядке, нужно разбить из на пары первый--последний, второй--предпоследний и т. д. и поменять местами элементы в каждой паре. При этом нужно обратить внимание, что пар не N, а [N/2].

ПРИМЕР КОДА НА ЯЗЫКЕ PYTHON

for i in range(len(A) // 2):
    A[i], A[N - i - 1] = A[N - i - 1], A[i]

На языке питон можно сделать то же самое короче, воспользовавшись срезами списков:
A = A[::-1]

ПРИМЕР КОДА НА ЯЗЫКЕ КУМИР

алг
нач
  цел n, i, tmp
  целтаб T[1:100]
  |считываем количество чисел в массиве
  ввод n
  |считываем сами числа
  нц для i от 1 до n
    ввод T[i]
  кц
  |переставим элементы в обратном порядке
  нц для i от 1 до div(n, 2)
    tmp:= T[i]
    T[i]:= T[n - i + 1]
    T[n - i + 1]:= tmp
  кц
  нц для i от 1 до n
    вывод T[i]
    вывод ' '
  кц
кон

ПРИМЕР КОДА НА ЯЗЫКЕ PASCAL

var
  a: array[1..100] of longint;
  tmp, i, n: longint;

begin
  readln(n); {считываем реальное количество элементов массива}
  for i := 1 to n do {считываем сами элементы массива}
    read(a[i]);

  for i := 1 to n div 2 do begin {меняем местами соответствующие элементы}
    tmp := a[i];
    a[i] := a[n - i + 1];
    a[n - i + 1] := tmp;
  end;

**for i := 1 to n do {выводим на экран элементы массива}
    write(a[i], ' ');**Циклический сдвиг в массиве

Информатика. Язык программирования Python в курсе информатики. Все классы.

Сдвиг влево

Двигаясь слева направо по массиву, присваиваем текущему элементу значение следующего.

Первый элемент нужно заблаговременно спасти во временной переменной, а затем положить его значение в конец массива.

ПРИМЕР НА PYTHON 3

tmp = A[0]
for i in range(len(A) - 1):
    A[i] = A[i+1]
A[len(A)-1] = tmp

ПРИМЕР НА PASCAL

tmp := A[0];
for i := 0 to N-2 do
  A[i] := A[i+1];
A[N-1] = tmp;

ПРИМЕР НА КУМИРЕ

алг
нач
  цел n, i, tmp
  целтаб T[1:100]
  |считываем количество чисел в массиве
  ввод n
  |считываем сами числа
  нц для i от 1 до n
    ввод T[i]
  кц
  |сдвинем элементы влево
  tmp := T[1]
  нц для i от 1 до n - 1
    T[i]:= T[i + 1]
  кц
  T[n]:= tmp
  нц для i от 1 до n
    вывод T[i]
    вывод ' '
  кц
кон

 Сдвиг вправо

Двигаясь по массиву, присваиваем текущему элементу значение соседнего с меньшим индексом. Только двигаться нужно **справа налево**.

Последний элемент нужно спасти во временной переменной, а затем положить его значение в начало массива.

ПРИМЕР НА PYTHON 3

tmp = A[-1]
for i in range(len(A)-1, 0, -1):
    A[i] = A[i-1]
A[0] = tmp

ПРИМЕР НА PASCAL

tmp := A[N-1];
for i := N-1 downto 1 do
  A[i] := A[i-1];
A[0] = tmp;

ПРИМЕР НА КУМИРЕ

алг
нач
  цел n, i, tmp
  целтаб T[1:100]
  |считываем количество чисел в массиве
  ввод n
  |считываем сами числа
  нц для i от 1 до n
    ввод T[i]
  кц
  |сдвинем элементы влево
  tmp := T[n]
  нц для i от n до 2 шаг -1
    T[i]:= T[i - 1]
  кц
  T[1]:= tmp
  нц для i от 1 до n
    вывод T[i]
    вывод ' '
  кц
кон

Дополнение

На языке Python для сдвига (неэффективного по памяти) можно воспользоваться срезами.

Сдвиг на k элементов влево:

A = A[k:] + A[:k]

Сдвиг на k элементов вправо:

A = A[-k:] + A[:-k]

Такой сдвиг требует дополнительную память для хранения срезов, пропорциональную длине массива.

Кортежи в Python

Информатика. Язык программирования Python в курсе информатики. Все классы.

Кортежи в Python

Кортеж — это неизменяемый список. Кортеж не может быть изменён никаким способом после его создания.

>>> a\_tuple = ("a", "b", "mpilgrim", "z", "example")  **(1)**

>>> a\_tuple

('a', 'b', 'mpilgrim', 'z', 'example')

>>> a\_tuple[0]                                        **(2)**

'a'

>>> a\_tuple[-1]                                       **(3)**

'example'

>>> a\_tuple[1:3]                                      (4)

('b', 'mpilgrim')

1. Кортеж определяется так же, как список, за исключением того, что набор элементов заключается в круглые скобки, а не в квадратные.
2. Элементы кортежа заданы в определённом порядке, как и в списке. Элементы кортежа индексируются с нуля, как и элементы списка, таким образом, первый элемент не пустого кортежа — это всегда a\_tuple[0].
3. Отрицательные значения индекса отсчитываются от конца кортежа, как и в списке. Последний элемент имеет индекс -1.
4. Создание среза кортежа («slicing») аналогично созданию среза списка. Когда создаётся срез списка, получается новый список; когда создаётся срез кортежа, получается новый кортеж.

Основное отличие между кортежами и списками состоит в том, что кортежи не могут быть изменены. На практике это означает, что у них нет методов, которые бы позволили их изменить. У списков есть такие методы, как append(), extend(), insert(), remove(), и pop(). У кортежей ни одного из этих методов нет. Можно взять срез от кортежа (так как при этом создастся новый кортеж).

# продолжение предыдущего примера

>>> a\_tuple ('a', 'b', 'mpilgrim', 'z', 'example')

>>> a\_tuple.append("new")               **(1)**

Traceback (innermost last):

File "<interactive input>", line 1, in ?

AttributeError: 'tuple' object has no attribute 'append'

>>> a\_tuple.remove("z")                 **(2)**

Traceback (innermost last):   File "<interactive input>", line 1, in ? AttributeError: 'tuple' object has no attribute 'remove'

>>> a\_tuple.index("example")            **(3)**

4

>>> "z" in a\_tuple                      **(4)**

True

*Перевод сообщений оболочки:*

Раскрутка стека (от внешних к внутренним):

Файл "<интерактивный ввод>", строка 1, позиция ?

AttributeError: у объекта 'tuple' нет атрибута '<атрибут>'

1. Вы не можете добавить элементы к кортежу. Кортежи не имеют методов append() или extend().
2. Вы не можете удалять элементы из кортежа. Кортежи не имеют методов remove() или pop().
3. Вы можете искать элементы в кортежи, поскольку это не изменяет кортеж.
4. Вы также можете использовать оператор in, чтобы проверить существует ли элемент в кортеже.

Так где же могут пригодиться кортежи?

* Кортежи в некоторых случаях быстрее, чем списки. Но такие оптимизации в каждом конкретном случае требуют дополнительных исследований.
* Кортежи делают код безопаснее в том случае, если у вас есть «защищенные от записи» данные, которые не должны изменяться.
* Некоторые кортежи могут использоваться в качестве элементов множества и ключей словаря (конкретно, кортежи, содержащие неизменяемые значения, например, строки, числа и другие кортежи). Списки никогда не могут использоваться в качестве ключей словаря, потому что списки — изменяемые объекты.

Кортежи могут быть преобразованы в списки и наоборот. Встроенная функция tuple() принимает список и возвращает кортеж из всех его элементов, функция list() принимает кортеж и возвращает список. По сути дела, tuple() замораживает список, а list() размораживает кортеж.

КОРТЕЖИ В ЛОГИЧЕСКОМ КОНТЕКСТЕ

Вы можете использовать кортежи в логическом контексте, например, в операторе if.

>>> def is\_it\_true(anything):

…   if anything:

…     print("да, это истина")

…   else:

…     print("нет, это ложь")

…

>>> is\_it\_true(())             (1)

нет, это ложь

>>> is\_it\_true(('a', 'b'))     (2)

да, это истина

>>> is\_it\_true((False,))       (3)

да, это истина

>>> type((False))              (4)

<class 'bool'>

>>> type((False,))

<class 'tuple'>

1. В логическом контексте пустой кортеж является ложью.
2. Любой кортеж состоящий по крайней мере из одного элемента — истина.
3. Любой кортеж состоящий по крайней мере из одного элемента — истина. Значения элементов не важны. Но что делает здесь эта запятая?
4. Чтобы создать кортеж из одного элемента, необходимо после него поставить запятую. Без запятой Python предполагает, что вы просто добавили еще одну пару скобок, что не делает ничего плохого, но и не создает кортеж.

ПРИСВАИВАНИЕ НЕСКОЛЬКИХ ЗНАЧЕНИЙ ЗА РАЗ

В Python можно использовать кортежи, чтобы присваивать значение нескольким переменным сразу.

>>> v = ('a', 2, True)

>>> (x, y, z) = v       (1)

>>> x

'a'

>>> y

2

>>> z

True

1. v — это кортеж из трех элементов, а (x, y, z) — кортеж из трёх переменных. Присвоение одного другому приводит к присвоению каждого значения из v каждой переменной в указанном порядке.

Это не единственный способ использования. Предположим, что вы хотите присвоить имена диапазону значений. Вы можете использовать встроенную функцию range() для быстрого присвоения сразу нескольких последовательных значений.

>>> (MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY) = range(7) >>> MONDAY

0

>>> TUESDAY

1

>>> SUNDAY

6

1. Встроенная функция range() создаёт последовательность целых чисел. (Строго говоря, функция range() возвращает итератор, а не список или кортеж) MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, и SUNDAY — определяемые переменные. (Этот пример заимствован из модуля calendar, небольшого забавного модуля, который отображает календари, примерно как программа cal из UNIX. В этом модуле определяются константы целого типа для дней недели.)
2. Теперь каждой переменной присвоено конкретное значение: MONDAY — это 0, TUESDAY — 1, и так далее.

При этом круглые скобки как в левой, так и в правой части присваивания можно опустить:

x, y = 0, 0

Вы также можете использовать присвоение значений нескольким переменным сразу, чтобы создавать функции, возвращающие несколько значений, для этого достаточно просто вернуть кортеж, содержащий эти значения. В том месте программы, где была вызвана функция, возвращаемое значение можно использовать как кортеж целиком, или присвоить значения нескольких отдельных переменных.