Занятие 7. Списки

1. 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 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:

запустить выполнить пошагово

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

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 в строке, 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(Primes) == 6`.

В отличие от строк, элементы списка можно изменять, присваивая им новые значения.

запустить выполнить пошагово

```
Rainbow = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Vi
print(Rainbow[0])
Rainbow[0] = 'красный'
print('Выведем радугу')
for i in range(len(Rainbow)):
    print(Rainbow[i])

```

Рассмотрим несколько способов создания и считывания списков. Прежде всего, можно создать пустой список (не содержащий элементов, длины 0), а в конец списка можно добавлять элементы при помощи метода `append`. Например, пусть программа получает на вход количество элементов в списке `n`, а потом `n` элементов списка по одному в отдельной строке. Вот пример входных данных в таком формате:

```
5
1809
1854
1860
1891
1925

```

В этом случае организовать считывание списка можно так:

запустить выполнить пошагово

```python
a = []  # заводим пустой список
n = int(input())  # считываем количество элемент в списке
for i in range(n):
    new_element = int(input())  # считываем очередной элемент
    a.append(new_element)  # добавляем его в список
    # последние две строки можно было заменить одной:
    # a.append(int(input()))
print(a)

```

В этом примере создается пустой список, далее считывается количество элементов в списке, затем по одному считываются элементы списка и добавляются в его конец. То же самое можно записать, сэкономив переменную `n`:

запустить выполнить пошагово

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

```

Для списков целиком определены следующие операции: конкатенация списков (сложение списков, т. е. приписывание к одному списку другого) и повторение списков (умножение списка на число). Например:

запустить выполнить пошагово

```
a = [1, 2, 3]
b = [4, 5]
c = a + b
d = b * 3
print([7, 8] + [9])
print([0, 1] * 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:

запустить выполнить пошагово

```
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:

запустить выполнить пошагово

```
a = [1, 2, 3, 4, 5]
for i in range(len(a)):
    print(a[i])

```

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

запустить выполнить пошагово

```
a = [1, 2, 3, 4, 5]
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'`.

> Обратите особое внимание на последний пример! Очень важная часть идеологии Питона — это цикл `for`, который предоставляет удобный способ перебрать все элементы некоторой последовательности. В этом отличие Питона от Паскаля, где вам обязательно надо перебирать именно *индексы* элементов, а не сами элементы.
> Последовательностями в Питоне являются строки, списки, значения функции range() (это не списки), и ещё кое-какие другие объекты.
>
> Приведем пример, демонстрирующий использование цикла `for` в ситуации, когда из строки надо выбрать все цифры и сложить их в массив как числа.

запустить выполнить пошагово

```
# дано: s = 'ab12c59p7dq'
# надо: извлечь цифры в список digits,
# чтобы стало так:
# digits == [1, 2, 5, 9, 7]

s = 'ab12c59p7dq'
digits = []
for symbol in s:
    if '1234567890'.find(symbol) != -1:
        digits.append(int(symbol))
print(digits)

```

2. Методы split и join

Элементы списка могут вводиться по одному в строке, в этом случае строку целиком можно считать функцией `input()`. После этого можно использовать метод строки `split()`, возвращающий список строк, которые получатся, если исходную строку разрезать на части по пробелам. Пример:

запустить выполнить пошагово

```
# на вход подаётся строка
# 1 2 3
s = input()  # s == '1 2 3'
a = s.split()  # a == ['1', '2', '3']

```

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:

запустить выполнить пошагово

```
a = input().split()
for i in range(len(a)):
    a[i] = int(a[i])

```

Используя специальную магию Питона — генераторы — то же самое можно сделать в одну строку:

запустить выполнить пошагово

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

```

Объяснение того, как работает этот код, будет дано в следующем разделе. Если нужно считать список действительных чисел, то нужно 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, вызов метода `split('.')` вернет список, полученный разрезанием исходной строки по символам `'.'`:

запустить выполнить пошагово

```
a = '192.168.0.1'.split('.')

```

В Питоне можно вывести список строк при помощи однострочной команды. Для этого используется метод строки `join`. У этого метода один параметр: список строк. В результате возвращается строка, полученная соединением элементов переданного списка в одну строку, при этом между элементами списка вставляется разделитель, равный той строке, к которой применяется метод. Мы знаем, что вы не поняли предыдущее предложение с первого раза. Поэтому смотрите примеры:

запустить выполнить пошагово

```
a = ['red', 'green', 'blue']
print(' '.join(a))
# вернёт red green blue
print(''.join(a))
# вернёт redgreenblue
print('***'.join(a))
# вернёт red***green***blue

```

Если же список состоит из чисел, то придется использовать еще тёмную магию генераторов. Вывести элементы списка чисел, разделяя их пробелами, можно так:

запустить выполнить пошагово

```
a = [1, 2, 3]
print(' '.join([str(i) for i in a]))
# следующая строка, к сожалению, вызывает ошибку:
# print(' '.join(a))

```

Впрочем, если вы не любитель тёмной магии, то вы можете достичь того же эффекта, используя цикл `for`.

3. Генераторы списков

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

запустить выполнить пошагово

```
n = 5
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 последовательность]

```

где *переменная* — идентификатор некоторой переменной, *последовательность* — последовательность значений, который принимает данная переменная (это может быть список, строка или объект, полученный при помощи `range`), *выражение* — некоторое выражение, как правило, зависящее от использованной в генераторе переменной, которым будут заполнены элементы списка.

Вот несколько примеров использования генераторов.

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

запустить выполнить пошагово

```
a = [0 for i in range(5)]

```

Создать список, заполненный квадратами целых чисел можно так:

запустить выполнить пошагово

```
n = 5
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)`:

запустить выполнить пошагово

```
n = 5
a = [i ** 2 for i in range(1, n + 1)]

```

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

запустить выполнить пошагово

```
1  from random import randrange
2  n = 10
3  a = [randrange(1, 10) for i in range(n)]
4
```

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:

запустить выполнить пошагово

```
1  a = [input() for i in range(int(input()))]
2
```

4. Срезы

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:

запустить выполнить пошагово

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

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

запустить выполнить пошагово

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

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` .

Обратите внимание, `A[i]` — это элемент списка, а не срез!

Операции со списками

Со списками можно легко делать много разных операций.

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

Ссылки на задачи доступны в меню слева. Эталонные решения теперь доступны на странице самой задачи.

Показать мои решения задач этого урока