Лабораторная работа №7
Списки

1 Цель работы

Изучить списки и научиться применять полученные знания на практике.

2 Краткая теория

2.1 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:

```
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 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.

```
Rainbow = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']
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:

```
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:

```
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:

```
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:

```
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:

```
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.

```
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.

```
# 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)
```

2.2 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:

```
# 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:

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

Используя генераторы, то же самое можно сделать в одну строку:

```
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 '.':

```
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:

```
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:

```
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.

2.3 Генераторы списков

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

```
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 функции 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:

```
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):

```
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):

```
from random import randrange
n = 10
a = [randrange(1, 10) 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()))]
```

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:

```
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, 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.

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

2.5 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.

3 Порядок выполнения работы

Получить задание для выполнения лабораторной работы (раздел 4) согласно своему варианту (по журналу). Разработать программу.

4 Задания для выполнения работы

Задание 1. Четные индексы.

Выведите все элементы списка с четными индексами (то есть A[0], A[2], A[4], ...).

Задание 2. Четные элементы.

Выведите все четные элементы списка. При этом используйте цикл for, перебирающий элементы списка, а не их индексы!

Задание 3. Больше предыдущего.

Дан список чисел. Выведите все элементы списка, которые больше предыдущего элемента.

Задание 4. Соседи одного знака.

Дан список чисел. Если в нем есть два соседних элемента одного знака, выведите эти числа. Если соседних элементов одного знака нет – не выводите ничего. Если таких пар соседей несколько – выведите первую пару.

Задание 5. Больше своих соседей.

Дан список чисел. Определите, сколько в этом списке элементов, которые больше двух своих соседей, и выведите количество таких элементов. Крайние элементы списка никогда не учитываются, поскольку у них недостаточно соседей.

Задание 6. Наибольший элемент.

Дан список чисел. Выведите значение наибольшего элемента в списке, а затем индекс этого элемента в списке. Если наибольших элементов несколько, выведите индекс первого из них.

Задание 7. Шеренга.

Петя перешёл в другую школу. На уроке физкультуры ему понадобилось определить своё место в строю. Помогите ему это сделать.

Программа получает на вход невозрастающую последовательность натуральных чисел, означающих рост каждого человека в строю. После этого вводится число X – рост Пети. Все числа во входных данных натуральные и не превышают 200.

Выведите номер, под которым Петя должен встать в строй. Если в строю есть люди с одинаковым ростом, таким же, как у Пети, то он должен встать после них.

Задание 8. Количество различных элементов.

Дан список, упорядоченный по неубыванию элементов в нем. Определите, сколько в нем различных элементов.

Задание 9. Переставить соседние.

Переставьте соседние элементы списка (A[0] с A[1], A[2] с A[3] и т. д.). Если элементов нечетное число, то последний элемент остается на своем месте.

Задание 10. Переставить min и max.

В списке все элементы различны. Поменяйте местами минимальный и максимальный элемент этого списка.

Задание 11. Удалить элемент.

Дан список из чисел и индекс элемента в списке k. Удалите из списка элемент с индексом k, сдвинув влево все элементы, стоящие правее элемента с индексом k.

Программа получает на вход список, затем число k. Программа сдвигает все элементы, а после этого удаляет последний элемент списка при помощи метода pop() без параметров.

Программа должна осуществлять сдвиг непосредственно в списке, а не делать это при выводе элементов. Также нельзя использовать дополнительный список. Также не следует использовать метод pop(k) с параметром.

Задание 12. Вставить элемент.

Дан список целых чисел, число k и значение C. Необходимо вставить в список на позицию с индексом k элемент, равный C, сдвинув все элементы, имевшие индекс не менее k, вправо.

Поскольку при этом количество элементов в списке увеличивается, после считывания списка в его конец нужно будет добавить новый элемент, используя метод append.

Вставку необходимо осуществлять уже в считанном списке, не делая этого при выводе и не создавая дополнительного списка.

Задание 13. Количество совпадающих пар.

Дан список чисел. Посчитайте, сколько в нем пар элементов, равных друг другу. Считается, что любые два элемента, равные друг другу образуют одну пару, которую необходимо посчитать.

Задание 14. Уникальные элементы.

Дан список. Выведите те его элементы, которые встречаются в списке только один раз. Элементы нужно выводить в том порядке, в котором они встречаются в списке.

Задание 15. Кегельбан.

N кеглей выставили в один ряд, занумеровав их слева направо числами от 1 до N. Затем по этому ряду бросили K шаров, при этом i-й шар сбил все кегли с номерами от l_i до r_i включительно. Определите, какие кегли остались стоять на месте.

Программа получает на вход количество кеглей N и количество бросков K. Далее идет K пар чисел l_i, r_i, при этом $1 \le l_i \le r_i \le N$.

Программа должна вывести последовательность из N символов, где j-й символ есть «I», если j-я кегля осталась стоять, или «.», если j-я кегля была сбита.

Задание 16. Ферзи.

Известно, что на доске 8×8 можно расставить 8 ферзей так, чтобы они не били друг друга. Вам дана расстановка 8 ферзей на доске, определите, есть ли среди них пара бьющих друг друга.

Программа получает на вход восемь пар чисел, каждое число от 1 до 8 – координаты 8 ферзей. Если ферзи не бьют друг друга, выведите слово NO, иначе выведите YES.