Тема 3.8 СПИСКИ

Большинство программ работает не с отдельными переменными, а с набором переменных. Для хранения таких данных можно использовать структуру данных, называемую в Python *список* (в большинстве же языков программирования используется другой термин «массив»).

Списки в Python (list) - упорядоченные изменяемые коллекции объектов произвольных типов.

В отличие от массивов, включающих в себя лишь однотипные элементы, списки не привязаны к определенной разновидности данных, а также не имеют жестких ограничений, связанных с их размером. Благодаря всем этим особенностям, списки являются достаточно гибким инструментом по работе с данными в Python.

Список представляет собой последовательность элементов, заключенных в квадратные скобки [ ],отделяющиеся друг от друга с помощью запятой, пронумерованных от 0, как символы в строке.

Создать список можно несколькими способами:

1. Получение списка через присваивание конкретных значений, используя конструкцию [ ]

```
l = [ ]                                  # это пустой список
l = [25, 755, -40, 57, -41]              # список целых чисел
l = [1.13, 5.34, 12.63, 4.6, 34.0, 12.8] # список из дробных чисел
l = ["Sveta", "Sergei", "Ivan", "Dasha"] # список из строк
l = ["Москва", "Иванов", 12, 124]        # смешанный список
l = [[0, 0, 0], [1, 0, 1], [1, 1, 0]]    # список, состоящий из списков
l = ['s', 'p', ['isok'], 2]              # список из значений и списка
```

2. Создание списка при помощи функции **List()**

```
l = list()              # пустой список
l = list ('spisok')     # 'spisok' - строка
print(l)                #['s', 'p', 'i', 's', 'o', 'k'] - результат - список
```

3. Создание списка при помощи функции **Split()**

```
stroka ="Hello, world"      # stroka - строка
l = stroka.split(",")       # l - список
print(l)                    # ['Hello', ' world']
```

4. Генераторы списков

```
# список из 10 элементов, заполненный единицами
l = [1]*10
# список l = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

l = [i for i in range(10)]
# список l = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

c = [c * 3 for c in 'list']
print (c) # ['lll', 'iii', 'sss', 'ttt']
```

from random import randint
l = [randint(10,80) for x in range(10)]
 # 10 чисел, сгенерированных случайным образом в диапазоне (10,80)

Каждый элемент списка имеет присвоенный ему индекс. Важно отметить, в Python индекс первого элемента в списке - 0.

z = [3, 7, 4, 2] # создаем список
 # обращение к первому элементу списка с индексом 0
print(z[0]) # 3

Python поддерживает отрицательную индексацию. Отрицательная индексация начинается с конца.

z = [3, 7, 4, 2] # создаем список
 # выведите последний элемент списка
print(z[-1]) # 2

Для того, чтобы изменить значение определенного элемента, используют оператор присваивания, обращаясь к элементу по индексу.

z = [3, 7, 4, 2] # создаем список
z[1] = 5 # замена 2 элемента списка
print(z) # [3, 5, 4, 2]

Срезы(slice) списка. Срезы хороши для получения подмножества значений списка.

z = [3, 7, 4, 2] # создаем список
print(z[0:2])   # Вывод элементов с индексом от 0 до 2 (не включая 2)
[3, 7]
print(z[:3])   # Все, кроме индекса 3
[3, 7, 4]
print(z[1:])   # начиная с индекса 1 до конца списка
[7, 4, 2]

Для ***ввода*** элементов списка используется цикл **for**:

L = [int(input()) for i in range(N)]

Функция *int* здесь используется для того, чтобы строка, введенная пользователем, преобразовывалась в целые числа.

Список можно ***выводить*** целым и поэлементно:

```
# вывод целого списка (массива)
print (L)
# поэлементный вывод списка (массива)
for i in range(N):
   print ( L[i], end = " " )
```

К спискам применяются ***операции соединения и повторения:***

1) Операция конкатенации:

l = [1, 3] + [4, 23] + [5]   # l = [1, 3, 4, 23, 5]

или

```
a=[33, -12, 'may']
b=[21, 48.5, 33]
print(a+b)                    # [33, -12, 'may', 21, 48.5, 33]
```

2) Операция повторения:

```
[[0,0],[0,1],[1,1]] * 2 # [[0, 0], [0, 1], [1, 1], [0, 0], [0, 1], [1, 1]]
```

У списков Python есть разные методы, которые помогают работать со списками (таблица 1).

Таблица 1 - Методы списков

Операция	Описание	Пример
x in a	Проверка, что x содержится в a	5 in [2, 3, 5]
x notin a	Проверка, что x не содержится в a То же, что и not (x in a)	5 notin [2, 3, 6]
a + a2	Конкатенация списков, то есть новый список, в котором сначала идут все элементы a, а затем все элементы a2	[2, 4] + [5, 3] == [2, 4, 5, 3]
a * k	Список a, повторенный k раз	[2, 3] * 3 == [2, 3, 2, 3, 2, 3]
a[n]	n-й элемент списка, отрицательные n — для отсчёта с конца	[2, 3, 7][0] == 2 [2, 3, 7][-1] == 7
a[start:stop:step]	Срез списка	[2, 3, 7][:2] == [2, 3]
len(a)	Длина списка	len([2, 3, 7]) == 3
max(a)	Максимальный элемент списка	max([2, 3, 7]) == 7
min(a)	Минимальный элемент списка	min([2, 3, 7]) == 2
sum(a)	Сумма элементов списка	sum([2, 3, 7]) == 12
a.index(x)	Индекс первого вхождения x в a (вызовет ошибку, если x notin a, то есть если x отсутствует в a)	[2, 3, 7].index(7) == 2
a.count(x)	Количество вхождений x в a	[2, 7, 3, 7].count(7) == 2
a.append(x)	Добавить x в конец a	a = [2, 3, 7] a.append(8) a == [2, 3, 7, 8]
a.extend(a2)	Добавить элементы коллекции a2 в конец a	a = [2, 3, 7] a.extend([8, 4, 5]) a == [2, 3, 7, 8, 4, 5]
del a[n]	Удалить n-й элемент списка	a = [2, 3, 7] del a[1] a == [2, 7]
Del a[start:stop:step]	Удалить из a все элементы, попавшие в срез	a = [2, 3, 7] del a[:2] a == [7]
a.clear()	Удалить из a все элементы (то же, что del a[:])	a.clear()
a.copy()	Копия a (то же, что и полный срез a[:])	b = a.copy()
a += a2 a *= k	Заменить содержимое списка на a+a2 и a*k соответственно	

a.insert(n, x)	Вставить x в a на позицию n, подвинув последующую часть дальше	a = [2, 3, 7] a.insert(0, 8) a == [8, 2, 3, 7]
a.pop(n)	Получить n-й элемент списка и одновременно удалить его из списка Вызов метода без аргументов равносилен удалению последнего элемента: a.pop() == a.pop(-1)	a = [2, 3, 7] a.pop(1) == 3 a == [2, 7]
a.remove(x)	Удалить первое вхождение x в a, в случае x notin a — ошибка	a = [2, 3, 7] a.remove(3) a == [2, 7]
a.reverse()	Изменить порядок элементов в a на обратный (перевернуть список)	a = [2, 3, 7] a.reverse() a == [7, 3, 2]
a.sort()	Отсортировать список по возрастанию	a = [3, 2, 7] a.sort() a == [2, 3, 7]
a.sort(reverse=True)	Отсортировать список по убыванию	a = [3, 2, 7] a.sort(reverse = True) a == [7, 3, 2]
bool(a)	Один из способов проверить список на пустоту (возвращает True, если список непустой, и False в противном случае)	

Двумерные списки

В задачах приходится хранить прямоугольные таблицы с данными. Базовые возможности платформы позволяют работать с двумерным представлением набора определенных значений в программе. В языке Python таблицу можно представить в виде списка строк, каждый элемент которого является в свою очередь списком, например, чисел.

Реализовать это можно при помощи обычного оператора присваивания, просто добавляя список в список Python для получения двумерного списка.

```
mass = [[1,2,3],
        [5,2,7],
        [7,7,9]] # получили матрицу 3x3
print (mass) # Выводит все списки в строку [[1,2,3],[5,2,7],[7,7,9]]
print (mass[1]) # Обращение ко второму списку [5,2,7]
print (mass[1][1]) # обращение ко второму элементу второго списка    [2]
```

Таким образом, можно увидеть, что двумерный список строится на нескольких одномерных.

Ввод двумерного списка

```
# в первой строке ввода идёт количество строк массива
n = int(input())
a = []
for i in range(n):
a.append([int(j) for j in input().split()])
```

Для ***обработки и вывода списка***, как правило, используют два вложенных цикла. Первый цикл перебирает номер строки, второй цикл бежит по элементам внутри строки.

Например, вывести двумерный числовой список на экран, разделяя числа пробелами внутри одной строки, можно так:

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j], end=' ')
print()
```

Сортировка списка «пузырьком»

```
n = int(input()) # количество элементов
a = []
for i in range(n): # считываем элементы списка
    a.append(int(input()))
# Сортировка пузырьком:
for i in range(n - 1):
    for j in range(n - 1 - i):
        if a[j] > a[j + 1]:
            a[j], a[j + 1] = a[j + 1], a[j]
print(a)
```