Список в Python -- это встроенный тип (класс) данных, представляющий собой одну из разновидностей структур данных. Структуру данных можно представить как сложную единицу, объединяющую в себе группу более простых. Каждая разновидность структур данных имеет свои особенности.

Отличительной особенностью списка является то, что он -- изменяемая последовательность элементов произвольных типов.

Давайте же объявим список a, содержащий в себе числа от 1 до 3 и строку «fox». Это делается очень просто:

a=[1, 2, 3, 'fox']

Помимо этого можно объявить пустой список или же с помощью конструктора класса:

s=[]

s=list(),

или же повторить один элемент несколько раз:

b=[0]*5

Пример выше вернет список из пяти нулей.

В Python список определяется квадратными скобками. Если в ходе работы нам потребуется использовать созданный список, достаточно обратиться к нему по имени – a.

При этом возможно обратиться к каждому элементу по отдельности. Пронумеруем все элементы, начиная с 0. Тогда элемент 1 будет под номером 0, 2 – под номером 1 и так далее. Обратиться к отдельному элементу можно следующим образом:

a[0]

Такая команда вернёт 1.

В Python существует также индексация с конца. Она начинается с -1:

a[-1]='fox'

Однако зачастую нам требуется взять не один элемент, а целый фрагмент массива. Такой фрагмент называется срезом. Для того, чтобы взять срез, нужно указать, элементы с какого по какой индекс нас интересуют.

a[1:3]=[2,3]

Заметим, что правая граница невключительна (точно так же, как в случае со строками).

Помимо этого, можно указать шаг, с которым будут выбираться значения. Например, если нас интересуют все значения с нечётными индексами, следует вывести все, начиная с первого с шагом 2.

a[1::2]= [2, 'fox']

Вы, конечно, заметили, что я не указала значение правой границы диапазона. Их допустимо оставлять пустыми, если подразумевается вывод с начала списка и/или до конца. Также шаг может быть отрицательным:

a[::-1] = ['fox', 3, 2, 1]

Однако важно помнить, что если левая граница меньше правой – шаг должен быть положительным, а если наоборот – отрицательным.

Часто говорят, что массив – это таблица. Однако пока мы рассматривали только таблицы, одна из размерностей которых равна единице. Это так называемые одномерные массивы. Однако возможны двух и даже трёхмерные массивы. Давайте рассмотрим примеры их объявлений.

На основании примера выше, есть желание объявить их как [[0]*9]*10. Однако при таком объявлении создаётся десять указателей на массив из 9 нулей. Таким образом, если что-то изменить в одной из строк, поменяются элементы этого столбца во всех строках.

Корректное объявление многомерных массивов неразрывно связано с методами списков. Методами называют функции, выполняющие задачи для определенного класса. В нашем случае, методы списков – функции, применяемые к спискам.

Как правило, почти всё, что нужно для комфортной работы с Python, уже реализовано в нём.

Рассмотрим некоторые из них:

- **append(item)**: добавляет элемент item в конец списка
- **insert(index, item)**: добавляет элемент item в список по индексу index
- **remove(item)**: удаляет элемент item. Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение ValueError
- **clear()**: удаление всех элементов из списка
- **index(item)**: возвращает индекс элемента item. Если элемент не найден, генерирует исключение ValueError
- **pop([index])**: удаляет и возвращает элемент по индексу index. Если индекс не передан, то просто удаляет последний элемент.
- **count(item)**: возвращает количество вхождений элемента item в список
- **sort([key])**: сортирует элементы. По умолчанию сортирует по возрастанию. Но с помощью параметра key мы можем передать функцию сортировки.
- **reverse()**: расставляет все элементы в списке в обратном порядке

Кроме того, Python предоставляет ряд встроенных функций для работы со списками:

- **len(list)**: возвращает длину списка
- **sorted(list, [key])**: возвращает отсортированный список
- **min(list)**: возвращает наименьший элемент списка
- **max(list)**: возвращает наибольший элемент списка

Вернёмся к многомерным спискам. Очевидное решение оказалось неправильным, однако всё ещё возможно создать вложенные списки таким образом:

a = [[1, 2, 3, 4], [5, 6, 7], [‘fox’, 8, 9]]

или так:

n = 5

```
m = 7
a = [0] * n
for i in range(n):
    a[i] = [0] * m
```

Помимо этого, можно воспользоваться так называемыми генераторами списков:

```
n = 5
m = 7
a = [[0] * m for i in range(n)]
```