Тема 6. Списки (list). Кортежи (tuple)

Списки (list)

Списки в Python – упорядоченные изменяемые коллекции объектов произвольных типов. В отличие от массива, тип объектов может быть различным. Аналогично массиву, элементы списка указываются в квадратных скобках []. Для того, чтобы работать со списком, его нужно задать.

6.1. Создание списка

Для **задания списка** используется **несколько способов**:

1. Можно обработать любой итерируемый объект встроенной функцией **list**.

```
c = list('Python')
print(c)
```

```
['P', 'y', 't', 'h', 'o', 'n']
```

2. Список можно задать при помощи литерала, т. е. заполнить его вручную.

```
c = []          # пустой список
c1 = [1, 2, 3] # список содержит 3 числа
print(c)
print(c1)
```

```
[]
[1, 2, 3]
```

3. Списки можно генерировать, используя циклы.
   Общий вид:

   <выражение> ***for*** *<переменная>* ***in*** *<последовательность>*

Пример 1: Заполнить список числами по порядку от 1 до 10.

```
s1 = [i for i in range(1, 11)]
print(s1)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Пример 2: Заполнить список 5 случайными числами из диапазона от 1 до 10 включительно.

```
import random
n = 5
s = [random.randint(1, 10) for i in range(n)]
print(s)
```

```
[7, 6, 5, 3, 1]
```

4. Списки можно заполнить с клавиатуры после запуска программы.

```
s1 = [int(input('Введите элемент списка: ')) for i in range(1, 11)]
print(s1)
```

В результате список заполнится введенными элементами и выведется на экран.

А вот так заполнить список с клавиатуры не получится:

```
s1 = []
for i in range(0, 10):
    s1[i] = int(input('введите элемент списка: '))
```

```
builtins.IndexError: list assignment index out of range
```

6.2. Обработка элементов списка

Все элементы списка нумеруются по порядку от 0 до *n*-1, где *n*-количество элементов в списке. В отличие от строк, в списках по индексу можно обращаться к элементам списка – как для получения его значения, так и для изменения. В строках по индексу можно получить только символ строки, а вот поменять его нельзя.

Получение значения элемента списка по его номеру

Получение элемента списка по номеру, аналогично работе с массивом в других языках программирования. Мы указываем имя списка, а в квадратных скобках его номер. Нумерация элементов списка начинается с 0.

```
s = [1, 2, 3, 4, 5]
print(s[1])
```

```
2
```

Данный способ работает не только для вывода. Допустим нам нужно сложить элементы списка, стоящие на нулевом и на втором месте.

```
s = [1, 2, 3, 4, 5]
a = s[0] + s[2]
print(a)
```

```
4
```

Также индекс может быть и отрицательным числом, тогда нумерация идет справа налево: -1 -2 -3…, т. е. начинается не с 0.

```
s = [1, 2, 3, 4]
print(s[0])
print(s[-1])
```

```
1
4
```

Срезы в списках

Срезы в списках позволяют также как и в строках, извлекать не по одному элементу, а группы элементов. Синтаксис точно такой же, как и в строках **list[start:end:step]**.

Пример: здесь результаты выполнения будут продемонстрированы в виде комментариев.

```
a = [1, 2, 3, 4, 5, 6, 7]
print(a[1])     # 2
print(a[1:])    # [2, 3, 4, 5, 6, 7]
print(a[1:4])   # [2, 3, 4]
print(a[1:5:2]) # [2, 4]
print(a[:3])    # [1, 2, 3]
```

Изменение значения элемента списка по его номеру

Изменение элемента списка по номеру работает аналогично изменению элемента массива в других языках программирования. Общий вид выглядит следующим образом:

список[<*номер элемента*>] = <*новое значение*>

```
s = [1, 2, 3, 4, 5]
print(s)
s[1] = 'Новый элемент'
print(s)
```

```
[1, 2, 3, 4, 5]
[1, 'Новый элемент', 3, 4, 5]
```

Заметьте, что изначально элемент был числовой, а мы заменили его на строковый. Тем самым показываем, что в списках могут храниться элементы разного типа.

6.3. Перебор элементов в списке

Перебор элементов списка, так же, как и его заполнение, можно производить при помощи цикла.

```
a = [1, 2, 5, 6, 7]
for i in a:
    print(i, end=' ')
```

```
1 2 5 6 7
```

Если необходимо перебрать все элементы списка и при этом изменить их, то здесь можно воспользоваться функцией **range**.

```
a = [1, 2, 5, 6, 7]
for i in range(len(a)):
    a[i] += 1
print(a)
```

```
[2, 3, 6, 7, 8]
```

Также элементы можно перебрать при помощи цикла **while**.

```
a = [1, 2, 5, 6, 7]
i = 0
b = len(a)
while (i < b):
    a[i] *= 2
    i += 1
print(a)
```

```
[2, 4, 10, 12, 14]
```

6.4. Функции и методы списков

Элементы списка можно обрабатывать стандартными встроенными функциями Python, такими как:

- **min() –** поиск минимального элемента.

```
a = [8, 2, 3, 2, 5]
print(min(a))
```

```
2
```

- **max() –** поиск максимального элемента.

```
a = [8, 2, 3, 2, 5]
print(max(a))
```

```
8
```

Стоит отметить, что поиск максимального и минимального элемента выполняется в случае, если все элементы списка одного общего типа, т. е. либо они все являются числами (неважно, целыми или дробными), либо все являются символами и т. д. Иначе интерпретатор выдаст ошибку.

Пример:

```
a = [1, 3, '0']
print(min(a))
```

```
builtins.TypeError: unorderable types: str() < int()
```

В результате произойдет ошибка сравнения числа и строки.

- **len() –** количество элементов в списке.

```
a = [8, 2, 3, 2, 5]
print(len(a))
```

```
5
```

- **list.append(***x***)** – добавляет элемент в конец списка.

```
a = [1, 2, 3]
a.append('list')
print(a)
```

```
[1, 2, 3, 'list']
```

- **list.extend(*L*) –** расширяет список *list*, добавляя в конец все элементы списка *L*.

```
a = [1, 2, 3]
b = [0, 5]
a.extend(b)
print(a)
```

```
[1, 2, 3, 0, 5]
```

- **list.insert(*i*, *x*) –** вставляет на *i*-ое место значение *x*, сдвигая все значения после него вперед (вправо).

```
a = [1, 2, 3]
a.insert(1, 'list')
print(a)
```

```
[1, 'list', 2, 3]
```

- **list.remove(*x*) –** удаляет первый элемент в списке, имеющий значение *x*.

```
a = [1, 2, 3, 2, 5]
print(a.remove(2))
print(a)
```

```
None
[1, 3, 2, 5]
```

- **list.pop([*i*]) –** удаляет *i*-ый элемент и возвращает его значение. Если индекс не указан, удаляется последний элемент.

```
a = [1, 2, 3, 2, 5]
print(a.pop(2))
print(a)
```

```
3
[1, 2, 2, 5]
```

- **list.index(*x*, [*start* [, *end*]]) –** возвращает положение первого элемента от *start* до *end*-1 со значением *x*

```
a = [1, 2, 3, 2, 5, 6, 2, 5, 7, 2]
print(a.index(2, 2, 4))
```

```
3
```

- **list.count(*x*) –** возвращает количество элементов со значением *x*.

```
a = [1, 2, 3, 2, 5, 6, 2, 5, 7, 2]
print(a.count(2))
```

```
4
```

- **list.sort(**[*key* = <*функция*>]**) –** сортирует список на основе функции. По умолчанию сортирует список в алфавитном порядке (для чисел – по возрастанию). *key* изучить отдельно.

```
a = [1, 2, 3, 2, 5, 6, 2, 5, 7, 2]
a.sort()
print(a)
```

```
[1, 2, 2, 2, 2, 3, 5, 5, 6, 7]
```

- **list.reverse() –** симметрично переворачивает список.

```
a = [1, 2, 3]
a.reverse()
print(a)
```

```
[3, 2, 1]
```

- **list.copy() –** поверхностная копия списка (создается новый составной объект со ссылками на объекты, находящиеся в оригинале).

```
a = [1, 2, 3]
b = a.copy()
print(b)
```

```
[1, 2, 3]
```

- **list.clear()** – очищает список.

```
a = [1, 2, 3]
a.clear()
print(a)
```

```
[]
```

Обратите внимание на то, что методы, предназначенные для изменения значения списка, меняют сам список, и результат выполнения метода не нужно присваивать какой-либо переменной:

```
a = [1, 4, 2, 3]
a.sort()
print(a)

a = a.reverse()
print(a)
```

```
[1, 2, 3, 4]
None
```

6.5. Поиск минимального (максимального) элемента в списке

Выше разбираются готовые функции для поиска минимума и максимума. Использование их не составляет никакого труда. Но смысл данного пункта в том, чтобы показать суть самого поиска, как алгоритма.

Поиск минимума и поиск максимума являются, по сути, аналогичными задачами. Разбирать их будем на примере поиска минимального элемента.

Алгоритм поиска:

1. Предполагаем, что минимальным является первый элемент.
2. Перебираем все элементы списка, начиная со второго, и сравниваем их с предполагаемым минимальным элементом. Если нашелся элемент меньше минимального, то присваиваем его на место минимального элемента.

Пример:

Заполнить список из 10 случайных целых чисел из диапазона от 0 до 10. Вывести список на экран. Найти минимальный элемент и вывести его отдельно.

```
import random # подключаем модуль для работы со случайными числами

# заполняем список десятью случайными числами
a = [random.randint(1, 10) for i in range(10)]
print(a) # выводим список на экран

# поиск минимального элемента
min = a[0] # предполагаем, что минимальный элемент – первый

for i in range(1, len(a)): # перебираем все оставшиеся элементы списка
    if a[i] < min: # сравниваем с минимальным
        min = a[i] # при необходимости переприсваиваем

print('Минимальный элемент списка: ', min) # выводим минимальный элемент на экран
```

6.6. Поиск указанного элемента в списке

Для поиска элемента в списке есть специальный метод, рассмотренный выше. Но наша задача разобрать, как искать элемент без использования этого метода. Это необходимо для понимания принципа работы со списками.

Алгоритм поиска:

Предполагаем, что искомого элемента в списке нет. При помощи цикла перебираем все элементы списка и сравниваем их с искомым элементом. Если нашли совпадение, то останавливаем цикл.

Пример:

Заполнить список 10 случайными целыми числами из диапазона от 0 до 10. Вывести список на экран. Запросить искомое число и ввести его с клавиатуры. Проверить, есть ли число в списке, и выдать соответствующее сообщение.

```
import random # подключаем модуль для работы со случайными числами

a = [random.randint(0, 10) for i in range(10)] # заполняем список десятью
                                               # случайными числами
print(a) # выводим список на экран
b = int(input('Введите целое число: ')) # запрашиваем и вводим искомое число

# поиск элемента
rez = False # предполагаем, что элемента нет в списке

for i in range(0, len(a)): # перебираем все элементы списка
    if a[i] == b:  # сравниваем с искомым элементом
        rez = True # если элемент найден, отмечаем это
        break      # останавливаем цикл

if rez == True: # если истина, то элемент есть, иначе элемента нет
    print('Элемент есть в списке')
else:
    print('Элемент в списке отсутствует')
```

6.7. Сортировка списка

Для сортировки списка также есть специальный метод **sort()**, но мы разберем сортировку без использования этого метода. Существует несколько способов сортировки элементов последовательностей. Здесь мы разберем только один – сортировку перестановкой.

Сортировка перестановкой подразумевает поочередное сравнение каждого элемента со всеми следующими за ним элементами и, при необходимости, перемену их местами. Этот вид сортировки очень медленный, но наиболее понятный.

```
import random

a = [random.randint(1, 10) for i in range(10)]
print(a)
for i in range(len(a)-1):
    for j in range(i+1, len(a)):
        if a[j] < a[i]:
            a[i], a[j] = a[j], a[i]
print(a)
```

6.8. Многомерные списки

В Python, кроме обычных одномерных списков, также используются и многомерные списки. Чаще всего это двумерные списки.

Многомерные списки также называют вложенными списками. Вложенными элементами могут быть и другие типы данных, например, кортежи, словари и т. п.

Многомерный список задается следующим образом:

$$a = [[], [], [], \dots , []]$$

Пример:

```
a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Исходя из правила, что выражение внутри скобок может быть записано в несколько строчек, удобно записать такой список в следующем виде:

```
a = [
      [1, 2, 3],
      [4, 5, 6],
      [7, 8, 9]
    ]
```

Обращение к элементам происходит таким образом: на первом месте указывается номер подсписка в главном списке, а затем номер непосредственно элемента в подсписке.

Пример:

```
a = [
      [1, 2, 3],
      [4, 5, 6],
      [7, 8, 9]
    ]

print(a[1][1])
print(a[0][2])
```

```
5
3
```

Также можно вывести не отдельный элемент, а один из вложенных списков.

```
a = [
      [1, 2, 3],
      [4, 5, 6],
      [7, 8, 9]
    ]

print(a[1])
```

```
[4, 5, 6]
```

Аналогично выводится весь двумерный список.

Пример:

```
a = [
      [1, 2, 3],
      [4, 5, 6],
      [7, 8, 9]
    ]

print(a)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Вывод двумерного списка в виде таблицы

При стандартном выводе двумерного списка все элементы выводятся в одну строчку, что может затруднять восприятие такого списка как многомерного. Гораздо удобнее, если список будет представлен в виде таблицы.

Для вывода в виде таблицы понадобится два вложенных цикла. Один цикл перебирает строки, а второй – элементы в каждой строке.

Допустим, у нас есть двумерный список, в котором 3 строки по 3 элемента. Зададим такой список в коде программы, а затем выведем его построчно.

```
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()
```

```
1 2 3
4 5 6
7 8 9
```

Заполнение двумерного списка

- **Заполнение двумерного списка вручную после запуска программы**

Заполнение двумерного списка после запуска программы не совсем тривиальная задача. При вводе элементов каждый вложенный список вводится с новой строки, а каждый элемент вложенного списка отделяется пробелом. В данном случае количество элементов во вложенных списках может быть разное.

```
n = int(input()) # указываем количество строк в списке
a = [] # создаем пустой список

for i in range(n):
    row = input().split() # считываем строку чисел через пробел,
                          # и разбиваем строку по разделителю,
                          # по умолчанию это пробел
# конвертируем элементы списка в числа (при необходимости)
    for i in range(len(row)):
        row[i] = int(row[i])
# добавляем список в качестве вложенного
    a.append(row)

print(a)
```

Пример результата выполнения программы:

```
2
1 3 5
2 4 6
[[1, 3, 5], [2, 4, 6]]
```

- **Заполнение двумерного списка случайными числами**

В данном примере рассмотрим заполнение элементов в случае, когда все вложенные списки имеют одинаковое количество элементов.

```
import random

n = int(input('Введите количество строк: '))
m = int(input('Введите количество элементов в строках: '))
a = [] # создаем пустой список
for i in range(n):
    row = [] # создаем пустой вложенный список
    for i in range(m):
        row.append(random.randint(1, 10))
# добавляем список в качестве вложенного
    a.append(row)

print(a)
```

Пример результата выполнения программы:

```
Введите количество строк: 3
Введите количество элементов в строках: 4
[[9, 10, 8, 4], [9, 3, 9, 5], [1, 4, 10, 9]]
```

- **Заполнение двумерного списка генератором**

Для создания двумерных списков можно использовать вложенные генераторы, разместив генератор списка, являющегося строкой, внутри генератора для строк. Например, сделать список из *n* строк и *m* столбцов при помощи генератора, создающего список из *n* элементов, каждый элемент которого является списком из *m* нулей:

[[0 for j in range(m)] for i in range(n)]

Но если число 0 заменить на некоторое выражение, зависящее от *i* (номер строки) и *j* (номер столбца), то можно получить список, заполненный по некоторой формуле.

Допустим, нужен список, в котором $n = 5$ строк, $m = 6$ столбцов, и элемент в строке *i* и столбце *j* вычисляется по формуле: $A[i][j] = i * j$.

Для создания такого массива можно использовать генератор:

*[[i * j for j in range(6)] for i in range(5)]*

Обработка элементов двумерного списка

Для обработки всех элементов двумерного списка нужен вложенный цикл. Принцип работы рассмотрим на примере нахождения суммы всех элементов двумерного списка.

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
s = 0
for i in range(len(a)):
    for j in range(len(a[i])):
        s += a[i][j]
print(s)
```

```
45
```

6.9. Кортежи (tuple)

Кортеж – это последовательность неизменяемых объектов. Кортежи очень схожи со списками. Разница заключается в том, что:

- элементы кортежа не могут быть изменены;
- используют обычные круглые скобки, вместо квадратных.

Преимущество кортежа по сравнению со списком в том, что он занимает меньше места в памяти.

Обработка элементов кортежа

Кортеж создается либо простым присваиванием элементов, разделенных запятыми, либо с указанием их в скобках, но тоже через запятую.

Пример:

```
a = ('кортеж', 'Python', 3, 0)
b = (1, 2, 3, 4)
c = 'a', 'b', 'c'
```

При необходимости создания пустого кортежа используют пустые скобки.

```
a = ()
```

Также кортеж можно создать из итерируемого объекта, например, из строки, используя встроенную функцию:

```
a = tuple('Python')
print(a)
```

```
('P', 'y', 't', 'h', 'o', 'n')
```

Заполнение кортежа с клавиатуры

```
s1 = tuple([int(input('Введите элемент массива: ')) for i in range(1,11)])
print(s1)
```

Элементы можно ввести с клавиатуры простым способом:

```
s = input('Введите по порядку, без пробелов, элементы кортежа: ')
# ввод: qwerty
a = tuple(s)
print(a)
```

```
('q', 'w', 'e', 'r', 't', 'y')
```

Заполнение кортежа случайными числами

```
import random
s1 = tuple([random.randint(1, 10) for i in range(1, 11)])
print(s1)
```

```
(1, 2, 7, 8, 4, 10, 2, 4, 9, 5)
```

Заполнение кортежа по правилу

Допустим, нам нужно заполнить кортеж из 10 элементов степенями двойки от 1 до 10.

```
s1 = tuple([pow(2, i) for i in range(1, 11)])
print(s1)
```

```
(2, 4, 8, 16, 32, 64, 128, 256, 512, 1024)
```

6.10. Получение данных из кортежа

Для того, чтобы получить какой-то элемент кортежа, необходимо обратиться к нему по индексу с указанием его в квадратных скобках [].

```
a = (1, 2, 3, 4)
print('a[1] =', a[1])
```

```
a[1] = 2
```

Если нужен диапазон элементов из кортежа, то пользуются срезом.

```
a = (1, 2, 3, 4)
print('a[1:3] =', a[1:3])
```

```
a[1:3] = (2, 3)
```

Если необходим весь кортеж, то к нему обращаемся просто по имени переменной.

```
a = (1, 2, 3, 4)
print('a =', a)
```

```
a = (1, 2, 3, 4)
```

Базовые операторы и методы кортежей

Над кортежами определены все те операции, которые определены над списками, но не изменяющие элементы списка.

Приведем некоторые в виде таблицы.

Функция, метод	Описание	Пример
len()	Длина кортежа (количество элементов).	a=(1,2,3,5,8) print(len(a)) **Получим:** 5

+	Объединение (склеивание) кортежей.	a=(1,2,5) b=(3,4) c=a+b print(c) **Получим:** (1,2,5,3,4)
in	Вхождение элемента в кортеж, если элемент присутствует, то получаем True, иначе False.	a=(1,2,3,4) print(2 in a) **Получим:** True
max()	Поиск максимального элемента кортежа. Работает только если все элементы кортежа однотипные.	a=(1,0,0.3) print(max(a)) **Получим:** 1 b=('f1', '3f', 'f2') print(max(b)) **Получим:** f2 a=(1,'0',0.3) print(max(a)) **Получим ошибку**
min()	Поиск минимального элемента кортежа. Работает только если все элементы кортежа однотипные.	a=(1,0,0.3) print(min(a)) **Получим:** 0 b=('f1', '3f', 'f2') print(min(b)) **Получим:** 3f a=(1,'0',0.3) print(min(a)) **Получим ошибку**

Обновление кортежа, удаление элемента

Изначально, как уже отмечалось выше, кортеж – это неизменяемая последовательность. Но все-таки иногда возникает необходимость его изменения, и тогда поступают следующим образом: создают новый кортеж с некоторыми элементами из старого и добавляют новые.

6.11. Функция zip

Функция **zip()** предназначена для объединения в кортежи элементов итерируемых последовательностей по следующему правилу. Первые элементы всех указанных в качестве аргументов последовательностей объединяются в первый кортеж, вторые элементы во второй кортеж и т. д. Процесс останавливается тогда, когда достигнут конец самой короткой последовательности. Как правило, для удобства дальнейшей работы все полученные кортежи объединяют в последовательность (список, кортеж, словарь, множество). Итерируемые последовательности, используемые в качестве аргументов в одной функции **zip(),** могут быть разного типа.

Пример:

```
a = 'строка'    #строка
b = [1, 'a', 2] #список
c = ('Кортеж', 'из', 'нескольких', 'слов') #кортеж
d = list(zip(a, b, c))   #получаем список кортежей
d1 = tuple(zip(a, b, c)) #получаем кортеж кортежей
print(d)
print(d1)
```

Результат:

```
[('с', 1, 'Кортеж'), ('т', 'а', 'из'), ('р', 2, 'нескольких')]
(('с', 1, 'Кортеж'), ('т', 'а', 'из'), ('р', 2, 'нескольких'))
```

6.12. Функция enumerate

Функция **enumerate()** применяется для упрощения перебора элементов итерируемых последовательностей, когда вместе с элементами требуется получить их индексы. Результатом работы функции является кортеж, состоящий из двух элементов – индекса элемента и его значения.

Пример:

```
a = [2, 4, 6]
for i in enumerate(a):
    print(i)
```

```
(0, 2)
(1, 4)
(2, 6)
```

При обработке всех элементов последовательности вместо комбинации **range(len())** может быть удобнее использовать функцию **enumerate()**.

Пример 1:

```
a = [2, 4, 6]
for i in range(len(a)):
    print(i, a[i])
```

```
0 2
1 4
2 6
```

Пример 2:

```
a = [2, 4, 6]
for i, val in enumerate(a):
    print(i, val)
```

```
0 2
1 4
2 6
```