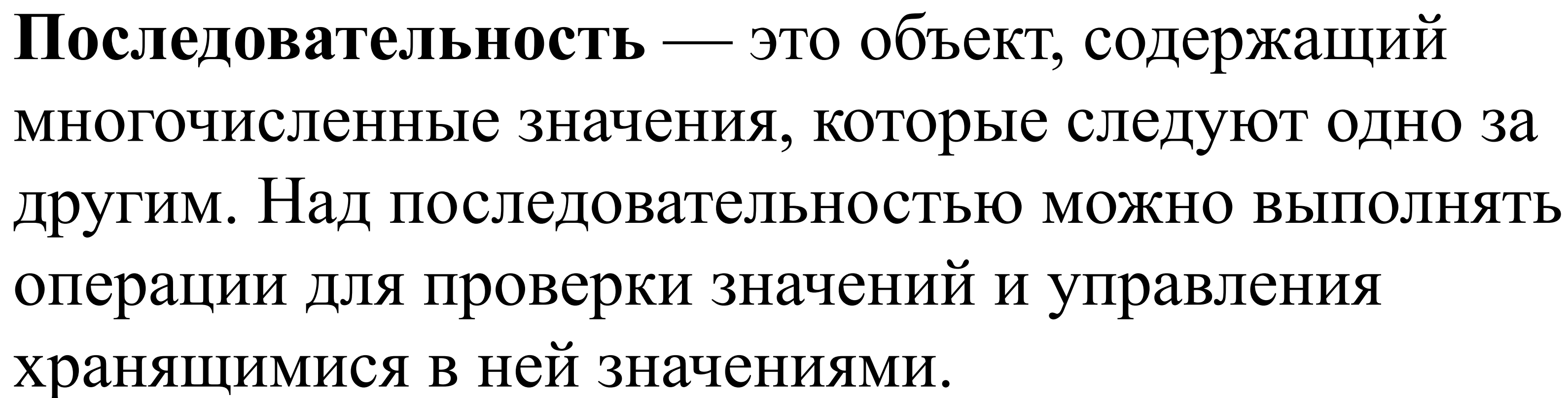
СПИСКИ В PYTHON


Последовательность — это объект, содержащий многочисленные значения, которые следуют одно за другим. Над последовательностью можно выполнять операции для проверки значений и управления хранящимися в ней значениями.

Последовательность — это объект в виде индексируемой коллекции, который содержит многочисленные значения данных. Хранящиеся в последовательности значения следуют одно за другим. Python предоставляет разнообразные способы выполнения операций над значениями последовательностей.

В Python имеется несколько разных типов объектов-последовательностей.

Списки и кортежи — это последовательности, которые могут содержать разные типы данных. Отличие списков от кортежей простое: список является мутирующей последовательностью (т. е. программа может изменять его содержимое), а кортеж — немутирующей последовательностью (т. е. после его создания его содержимое изменить невозможно).

Список— это объект, который содержит многочисленные элементы данных.
Списки являются мутирующими последовательностями, т. е. их содержимое может изменяться во время выполнения программы. Списки являются динамическими структурами данных, т. е. элементы в них могут добавляться и удаляться из них.
Для работы со списками в программе можно применять индексацию, нарезку и другие разнообразные методы.
Список — это объект, который содержит многочисленные элементы данных. Каждая хранящаяся в списке порция данных называется *элементом.*
even_numbers = [2, 4, 6, 8, 10]
Значения, заключенные в скобки и отделенные запятыми, являются элементами списка. После исполнения приведенной выше инструкции переменная even_numbers будет ссылаться на список.

Для работы с наборами данных Python предоставляет встроенные типы как списки, кортежи и словари.
такие

Список (list) представляет тип данных, который хранит набор или последовательность элементов. Для создания списка в квадратных скобках ([]) через запятую перечисляются все его элементы.

Во многих языках программирования есть аналогичная структура данных, которая называется массив.

Пример

```
1  numbers = [1, 2, 3, 4, 5]
```

Также для создания списка можно использовать конструктор **list()**:

```
1  numbers1 = []
2  numbers2 = list()
```

Оба этих определения списка аналогичны - они создают пустой список.

Индексы

Для обращения к элементам списка надо использовать индексы, которые представляют номер элемента в списка. Индексы начинаются с нуля. То есть второй элемент будет иметь индекс 1. Для обращения к элементам с конца можно использовать отрицательные индексы, начиная с -1. То есть у последнего элемента будет индекс -1, у предпоследнего - -2 и так далее.

```
1  numbers = [1, 2, 3, 4, 5]
2  print(numbers[0])   # 1
3  print(numbers[2])   # 3
4  print(numbers[-3])  # 3
5
6  numbers[0] = 125  # изменяем первый элемент списка
7  print(numbers[0])   # 125
```

Оператор повторения

Символ * перемножает два числа. Однако когда операнд слева от символа * является последовательностью (в частности, списком), а операнд справа — целым числом, он становится ***<u>оператором повторения</u>***. Оператор повторения делает многочисленные копии списка и все их объединяет. Вот общий формат операции:

*СПИСОК * n*

В данном формате *список* — это обрабатываемый список, *n* — число создаваемых копий.

```
>>> numbers = [0] * 5
>>> print(numbers)
[0, 0, 0, 0, 0]
>>>
```

Если необходимо создать список, в котором повторяется одно и то же значение несколько раз, то можно использовать символ звездочки *.

Например, определим список из шести пятерок:

```
numbers = [5] * 6  # [5, 5, 5, 5, 5, 5]
print(numbers)
```

Кроме того, если нам необходим список чисел, последовательности, то для его создания удобно использовать функцию range, которая имеет три формы:

- **range(end):** создается набор чисел от 0 до числа end
- **range(start, end):** создается набор чисел от числа start до числа end
- **range(start, end, step):** создается набор чисел от числа start до числа end с шагом step

```
numbers = list(range(10))
print(numbers)      # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
numbers = list(range(2, 10))
print(numbers)      # [2, 3, 4, 5, 6, 7, 8, 9]
numbers = list(range(10, 2, -2))
print(numbers)      # [10, 8, 6, 4]
```

Например, следующие два определения списка будут аналогичны, но за счет функции range мы сокращаем объем кода:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
numbers2 = list(range(1, 10))
```

Список необязательно должен содержать только однотипные объекты. Мы можем поместить в один и тот же список одновременно строки, числа, объекты других типов данных:

```
objects = [1, 2.6, "Hello", True]
```

Перебор элементов

Для перебора элементов можно использовать как цикл for, так и цикл while.

Перебор с помощью цикла for:

```
companies = ["Microsoft", "Google", "Oracle", "Apple"]
for item in companies:
    print(item)
```

Здесь вместо функции **range** мы сразу можем подставить имеющийся список **companies.**

Методы и функции по работе со списками

Для управления элементами списки имеют целый ряд методов.

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

Методы и функции по работе со списками

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

Добавление и удаление элементов

Для добавления элемента применяются методы **append()** **и** **insert()**, а для удаления - методы **remove(), pop() и clear().**

Добавление и удаление элементов

```
users = ["Tom", "Bob"]

# добавляем в конец списка
users.append("Alice")  # ["Tom", "Bob", "Alice"]
# добавляем на вторую позицию
users.insert(1, "Bill")          # ["Tom", "Bill", "Bob", "Alice"]

# получаем индекс элемента
i = users.index("Tom")
# удаляем по этому индексу
removed_item = users.pop(i)            # ["Bill", "Bob", "Alice"]

last_user = users[-1]
# удаляем последний элемент
users.remove(last_user)           # ["Bill", "Bob"]

print(users)

# удаляем все элементы
users.clear()
```

Проверка наличия элемента

Если определенный элемент не найден, то методы **remove и index** генерируют исключение. Чтобы избежать подобной ситуации, перед операцией с элементом можно проверять его наличие с помощью ключевого слова **in:**

```
companies = ["Microsoft", "Google", "Oracle", "Apple"]
item = "Oracle"  # элемент для удаления
if item in companies:
    companies.remove(item)

print(companies)
```

Выражение **item in companies** возвращает **True**, если элемент **item** имеется в списке **companies.** Поэтому конструкция **if item in companies** может выполнить последующий блок инструкций в зависимости от наличия элемента в списке.

Подсчет вхождений

Если необходимо узнать, сколько раз в списке присутствует тот или иной элемент, то можно применить метод **count():**

```
users = ["Tom", "Bob", "Alice", "Tom", "Bill", "Tom"]

users_count = users.count("Tom")
print(users_count)      # 3
```

Сортировка

Для сортировки по возрастанию применяется метод sort():

```
users = ["Tom", "Bob", "Alice", "Sam", "Bill"]

users.sort()
print(users)      # ["Alice", "Bill", "Bob", "Sam", "Tom"]
```

Если необходимо отсортировать данные в обратном порядке, то мы можем после сортировки применить метод `reverse()`:

```
users = ["Tom", "Bob", "Alice", "Sam", "Bill"]

users.sort()
users.reverse()
print(users)      # ["Tom", "Sam", "Bob", "Bill", "Alice"]
```

Минимальное и максимальное значения

Встроенные функции Python **min()** и **max()** позволяют найти минимальное и максимальное значения соответственно:

```
1 numbers = [9, 21, 12, 1, 3, 15, 18]
2 print(min(numbers))     # 1
3 print(max(numbers))     # 21
```

Пример:

В программе ниже приведен пример присвоения элементам списка вводимых пользователем значений. Пользователю нужно ввести суммы продаж, которые будут присвоены списку.

```
# Константа NUM_DAYS содержит количество дней,
# за которые мы соберем данные продаж.
NUM_DAYS = 5

# Создать список, который будет содержать
# продажи за каждый день.
sales = [0] * NUM_DAYS

# Создать переменную, которая будет содержать индекс.
index = 0

print('Введите продажи за каждый день.')

# Получить продажи за каждый день.
while index < NUM_DAYS:
      print('День № ', index +1,': ', sep='', end='')
      sales[index] = float (input ())
      index += 1

# Показать введенные значения.
print('Вот значения, которые были введены:')
for value in sales:
      print(value)
```

Вывод программы:

```
Введите продажи за каждый день.
День № 1: 10
День № 2: 20
День № 3: 30
День № 4: 40
День № 5: 50
Вот значения, которые были введены:
10.0
20.0
30.0
40.0
50.0
>>>
```

КОПИРОВАНИЕ СПИСКОВ

При копировании списков следует учитывать, что списки представляют изменяемый тип, поэтому если обе переменных будут указывать на один и тот же список, то изменение одной переменной, затронет и другую переменную:

```
1 people1 = ["Tom", "Bob", "Alice"]
2 people2 = people1
3 people2.append("Sam")   # добавляем элемент во второй список
4 # people1 и people2 указывают на один и тот же список
5 print(people1)   # ["Tom", "Bob", "Alice", "Sam"]
6 print(people2)   # ["Tom", "Bob", "Alice", "Sam"]
```

КОПИРОВАНИЕ СПИСКОВ

Чтобы происходило копирование элементов, но при этом переменные указывали на разные списки, необходимо выполнить глубокое копирование. Для этого можно использовать метод copy():

```
people1 = ["Tom", "Bob", "Alice"]
people2 = people1.copy()    # копируем элементы из people1 в people2
people2.append("Sam")   # добавляем элемент ТОЛЬКО во второй список
# people1 и people2 указывают на разные списки
print(people1)   # ["Tom", "Bob", "Alice"]
print(people2)   # ["Tom", "Bob", "Alice", "Sam"]
```

ПРИМЕРЫ:

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.

```
1  a = []  # заводим пустой список
2  n = int(input())  # считываем количество элемент в списке
3  for i in range(n):
4      new_element = int(input())  # считываем очередной элемент
5      a.append(new_element)  # добавляем его в список
6      # последние две строки можно было заменить одной:
7      # a.append(int(input()))
8  print(a)
9
```

ПРИМЕР
Ы:

То же самое можно записать, сэкономив переменную n:

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

Конкатенация списков:

Конкатенировать означает соединить две части воедино. Для конкатенации двух списков используется оператор +. Вот пример:

```
List1 = [1, 2, 3, 4]

list2 = [5, 6, 7, 8]

list3 = list1 + list2
```

После исполнения этого фрагмента кода списки list1 и list2 останутся неизменными, а list3 будет ссылаться на такой список:

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

Конкатенация списков:

```
>>> girl_names = ['Джоанна', 'Карен', 'Лори']
>>> boy_names = ['Крис', 'Джерри', 'Уилл']
>>> all_names = girl_names + boy_names
>>> print(all_names)
['Джоанна', 'Карен', 'Лори', 'Крис', 'Джерри', 'Уилл']
>>> 
```

Для того чтобы соединить два списка, можно воспользоваться расширенным оператором присваивания +=. Вот пример:

list1 = [1, 2, 3, 4]

list2 = [5, 6, 7, 8]

list1 += list2

Последняя инструкция присоединяет список list2 к списку list1. После исполнения этого фрагмента кода list2 останется неизменным, но list1 будет ссылаться на список:

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

Нарезка списка

Выражение среза извлекает диапазон элементов из последовательности.

Ранее было показано, каким образом индексация позволяет извлекать конкретный элемент из последовательности. Иногда возникает необходимость извлечь из последовательности более одного элемента. В Python можно составлять выражения, которые извлекают части последовательности, которые называются срезами.

Срез — это диапазон элементов, которые извлекаются из последовательности. Для того чтобы получить срез списка, пишут выражение в приведенном ниже общем формате:

имя_ *списка [начало : конец: шаг]*

В данном формате *начало*— это индекс первого элемента в срезе, *конец*— индекс, отмечающий конец среза, шаг – определяет через какое значение производить срез, если не указано, то значение 1. Это выражение возвращает список, содержащий копию элементов с *начала* до *конца* (но не включая последний).

Пример:

days = ['понедельник', 'вторник', 'среда', 'четверг', 'пятница’, 'суббота',’воскресенье']

Приведенная ниже инструкция применяет выражение среза для получения элементов, начиная с индекса 2 вплоть до, но не включая, 5:

```
mid_days = days[2:5]
```

После исполнения этой инструкции переменная mid_days будет ссылаться на приведенный ниже список:

['Среда', 'Четверг', 'Пятница']

Если в выражении среза опустить индекс *начало,* то Python будет использовать 0 в качестве начального значения индекса. Приведенный ниже сеанс интерактивного режима демонстрирует пример:

```
>>> numbers = [1, 2, 3, 4, 5]
>>> numbers[1:3]
[2, 3]
>>> numbers[0:3]
[1, 2, 3]
>>> numbers[1:]
[2, 3, 4, 5]
>>> numbers[:3]
[1, 2, 3]
>>> 
```

Обратите внимание, что строка 6 отправляет срез numbers [1:] в функцию print в качестве аргумента. Поскольку конечное значение индекса было опущено, срез содержит элементы, начиная с индекса 1 и заканчивая последним индексом, совпадающим с длинной списка. Аналогично если упустить начальный список, то в качестве начального индекса будет

Если в выражении среза опустить сразу оба индекса *начала* и *конца,* то получится копия всего списка.

Для получения среза можно использовать отрицательную индексацию, понимая, что последний индекс это -1

```
>>> numbers = [1, 2, 3, 4, 5]
>>> numbers[-3:-1]
[3, 4]
>>>
```

Рассмотрим случаи использования шага среза:

```
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> numbers[1:5:2]
[2, 4]
>>>
```

Получение всех четных элементов, обратите внимание что конечный индекс упустили:

```
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> numbers[1::2]
[2, 4, 6, 8, 10]
>>>
```

Список в обратном порядке:

```
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> numbers[-1::-1]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>>
```

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.

```
1  # на вход подаётся строка
2  # 1 2 3
3  s = input()  # s == '1 2 3'
4  a = s.split()  # a == ['1', '2', '3']
5
```

МЕТОДЫ SPLIT И JOIN

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.

```
1  a = ['red', 'green', 'blue']
2  print(' '.join(a))
3  # вернёт red green blue
4  print(''.join(a))
5  # вернёт redgreenblue
6  print('***'.join(a))
7  # вернёт red***green***blue
8
```

ГЕНЕРАТОРЫ СПИСКОВ

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

```
n = 5
a = [0] * n
```

ГЕНЕРАТОРЫ СПИСКОВ

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.

ПРИМЕ
Р

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

запустить выполнить пошагово

```
1 a = [0 for i in range(5)]
2
```

Создать список, заполненный квадратами целых чисел можно так:

запустить выполнить пошагово

```
1 n = 5
2 a = [i ** 2 for i in range(n)]
3
```

Если нужно заполнить список квадратами чисел от 1 до `n`, то можно изменить параметры функции `range` на `range(1, n + 1)`:

запустить выполнить пошагово

```
1 n = 5
2 a = [i ** 2 for i in range(1, n + 1)]
3
```

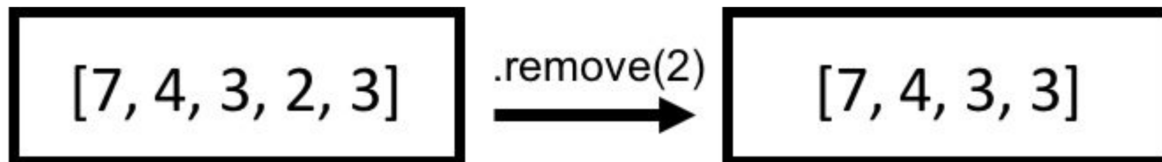
ПРИМЕР:

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

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

МЕТОД REMOVE

Метод remove удаляет первое вхождение значения в списке.

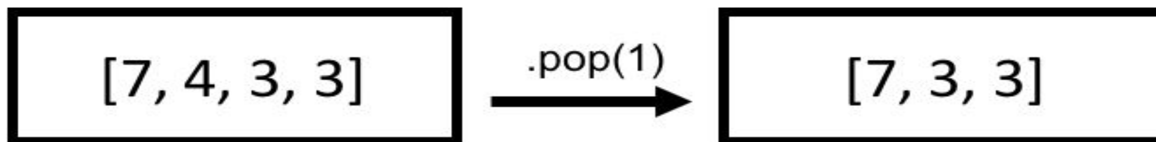


КОПИРОВАТЬ

```
z = [7, 4, 3, 2, 3]
z.remove(2)
print(z)
```

МЕТОД POP

z.pop(1) удаляет значение в индексе 1 и возвращает значение 4



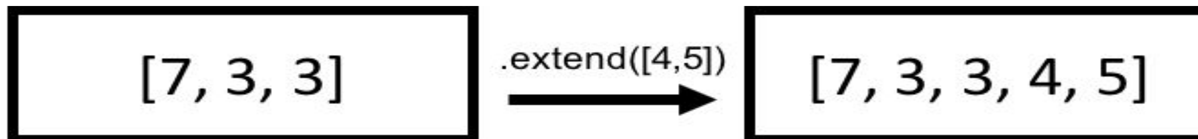
z.pop(1) удаляет значение в индексе 1 и возвращает значение 4

Метод `pop` удаляет элемент в указанном индексе. Этот метод также вернет элемент, который был удален из списка. В случае, если вы не указали индекс, он по умолчанию удалит элемент по последнему индексу.

КОПИРОВАТЬ

```
z = [7, 4, 3, 3]
print(z.pop(1))
print(z)
```

МЕТОД EXTEND



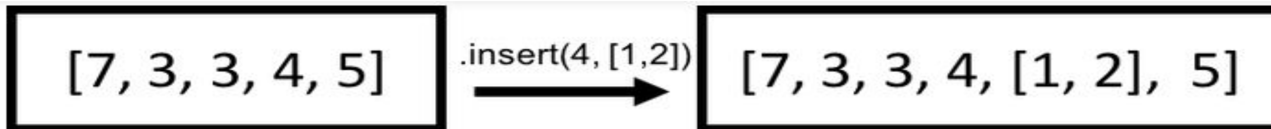
Метод `extend` расширяет список, добавляя элементы. Преимущество над `append` в том, что вы можете добавлять списки.
Добавим `[4, 5]` в конец `z`:

КОПИРОВАТЬ

```
z = [7, 3, 3]
z.extend([4,5])
print(z)
```

```
[7, 3, 3, 4, 5]
```

МЕТОД INSERT



Вставляет [1,2] с индексом 4

Метод `insert` вставляет элемент перед указанным индексом.

КОПИРОВАТЬ

```
z = [7, 3, 3, 4, 5]
z.insert(4, [1, 2])
print(z)
```

```
[7, 3, 3, 4, [1, 2], 5]
```

Оператор in и not in

Оператор `in` проверяет находится ли элемент в списке. При успешном результате он возвращает `True`, в случае неудачи, возвращает `False`.

КОПИРОВАТЬ

```
>>> list1 = [11, 22, 44, 16, 77, 98]
>>> 22 in list1
True
```

Аналогично `not in` возвращает противоположный от оператора `in` результат.

КОПИРОВАТЬ

```
>>> 22 not in list1
False
```

ИТЕРАЦИЯ ПО СПИСКУ С ИСПОЛЬЗОВАНИЕМ ЦИКЛА FOR

Список — последовательность. Ниже способ, которые вы можете использовать для цикла, чтобы перебрать все элементы списка.

КОПИРОВАТЬ

```
list1 = [1,2,3,4,5]
for i in list1:
    print(i, end=" ")
```

```
1 2 3 4 5
```