* [ОБЗОР КУРСА](https://lyceum.yandex.ru/courses/123/groups/568)

[Урок Списки](https://lyceum.yandex.ru/courses/123/groups/568/lessons/675)

**Знакомство со списками**

**План урока**

1

[Списки](https://lyceum.yandex.ru/courses/123/groups/568/lessons/675/materials/976#1)

2

[Создание списков](https://lyceum.yandex.ru/courses/123/groups/568/lessons/675/materials/976#2)

3

[Индексация в списках](https://lyceum.yandex.ru/courses/123/groups/568/lessons/675/materials/976#3)

4

[Добавление элемента в список](https://lyceum.yandex.ru/courses/123/groups/568/lessons/675/materials/976#4)

5

[Перебор элементов списка](https://lyceum.yandex.ru/courses/123/groups/568/lessons/675/materials/976#5)

6

[Срезы списков](https://lyceum.yandex.ru/courses/123/groups/568/lessons/675/materials/976#6)

7

[Списки и массивы](https://lyceum.yandex.ru/courses/123/groups/568/lessons/675/materials/976#7)

**Аннотация**

*В уроке рассматривается новый тип данных — списки (list), обращение к элементам списка по индексу (аналогично строкам, но с возможностью изменения элементов списка) и метод append. А также вопросы перебора элементов списка и срезов списка.*

**1. Списки**

Мы уже знаем тип данных, который называется **множество** и является **коллекцией (контейнером)**, то есть позволяет хранить несколько элементов данных, и тип строка, который тоже обладает свойствами коллекции. Сегодня мы познакомимся с ещё одним типом-коллекцией, который называется **список** (list). Никогда не создавайте переменные с таким именем!

**Списки**

Списки являются очень гибкой структурой данных и широко используются в программах. Давайте рассмотрим основные свойства списка в сравнении с теми коллекциями, которые мы уже знаем:

— Список хранит несколько элементов под одним именем (*как и множество*)

— Элементы списка могут повторяться (*в отличие от множества*)

— Элементы списка упорядочены и проиндексированы, доступна операция среза (*как в строке*)

— Элементы списка можно изменять (*в отличие от строки*)

— Элементами списка могут быть значения любого типа: целые и действительные числа, строки и даже другие списки

**2. Создание списков**

Чтобы работать со списком, нужно его создать. Для создания списка используется операция присваивания.

**Важно**

Чтобы задать готовый список, нужно справа от знака присваивания в квадратных скобках перечислить его элементы через запятую. Здесь создаётся список из первых пяти простых чисел, который помещается в переменную primes («простые числа»):

primes = [2, 3, 5, 7, 11]

**print**(primes) *# выводим на экран список целиком*

Для того, чтобы создать пустой список, можно воспользоваться конструкцией [] или функцией **list**.

empt1 = [] *# это пустой список*

empt2 = list() *# и это тоже пустой список*

Теперь вспомним один из рассмотренных нами ранее способов создания cтроки из заданного количества повторяющихся подстрок. Такую строку можно легко составить путём умножения на число:

**print**("'$' \* 5 -->", '$' \* 5)

**print**("'-|' \* 4 + '-' -->", '-|' \* 4 + '-')

Получим:

'$' \* 5 --> $$$$$

'-|' \* 4 + '-' --> -|-|-|-|-

Аналогично поступают и со списками:

**print**([2, 3] \* 4)

[2, 3, 2, 3, 2, 3, 2, 3]

**Важно**

Для генерации списков и строк, состоящих строго из повторяющихся элементов (например, список из заданного количества нулей), умножение на число — самый короткий и красивый метод.

my\_list = [0] \* 4

**3. Индексация в списках**

Чтобы получить отдельный элемент списка, нужно записать после него (или имени переменой, связанной с данным списком) в квадратных скобках номер (индекс) нужного элемента. Индекс отсчитывается с нуля — как в строках. Так же, как и в строках, для нумерации с конца разрешены отрицательные индексы.

Таким образом, мы умеем использовать квадратные скобки в Python уже для трех вещей: задания нового списка (тогда внутри этих скобок перечисляются его элементы), получения элемента списка или строки по индексу (тогда в скобках указывается индекс) и получения среза строки.

**print**('Сумма первых двух простых чисел:', primes[0] + primes[1])

**print**('Последнее из простых чисел в нашем списке:', primes[-1])

Как и в строках, попытка обратиться к индексу с несуществующим индексом вызовет ошибку:

**print**(primes[5]) *# ошибка: index out of range*

**4. Добавление элемента в список**

Добавление элемента в конец списка делается при помощи метода **append** (этот метод аналогичен методу add, используемому для добавления элементов в множество):

primes.append(13)

primes.append(15) *# ой, ошиблись -- 15 составное число!*

Обратите внимание: для того, чтобы воспользоваться методом append, нужно, чтобы список был создан (при этом он может быть пустым).

**Важно**

Попытка применить метод append к несуществующему списку приведет к ошибке. Заметьте, что два вызова метода append в следующем примере добавляют элемент к двум разным спискам.

odd\_numbers = [1, 3, 5, 7, 9, 11, 13, 15, 17]

primes.append(19)

odd\_numbers.append(19)

Кроме того вы можете расширить имеющийся список любым итерабельным (перечисляемым) объектом с помощью метода **extend**:

my\_list = [1, 2, 3]

another\_list = [4, 5, 6]

my\_list.extend(another\_list)

**print**(my\_list)

Получим:

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

**Важно**

Имейте в виду, что строка является итерируемой, поэтому, если вы расширите список со строкой, вы добавите каждый символ, когда будете перебирать строку (что может быть не так, как вы хотите):

my\_list = [1, 2, 3]

another\_list = "привет"

my\_list.extend(another\_list)

**print**(my\_list) *# [1, 2, 3, 'п', 'р', 'и', 'в', 'е', 'т']*

**Важно**

В отличие от отдельных символов в строках, элемент списка можно поместить справа от «=» в операторе присваивания и тем самым изменить этот элемент:

primes[6] = 17 *# Исправляем ошибку:*

*# седьмое (нумерация элементов списка - с нуля!)*

*# простое число - не 15, а 17.*

Тем не менее, многие вещи, которые можно делать со строками, можно делать и со списками:

**print**(len(primes)) *# выводим длину списка*

primes += [23, 29] *# списки можно складывать, как и строки*

**print**(primes) *# выведет [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]*

**if** 1 **in** primes: *# можно проверять, содержится ли в списке элемент*

**print**('Мы считаем единицу простым числом.')

**else**:

**print**('Мы, как и всё остальное человечество,   
 не считаем 1 простым числом.')

**5. Перебор элементов списка**

Во время выполнения программы текущее количество элементов списка всегда известно. Поэтому если нужно что-то сделать с каждым элементом списка (например, напечатать его на экране), можно перебрать элементы с помощью цикла for. При этом, как и для строк, возможны два варианта перебора: перебор индексов и перебор самих элементов.

**for** i **in** range(len(primes)):

*# выведем по очереди все элементы списка...*

**print**('Простое число номер', i + 1, '-', primes[i])

**for** p **in** primes:

**print**('Квадрат числа', p, '-', p \*\* 2) *# и их квадраты*

Заметьте, что при использовании конструкции for i in range(len(имя\_списка)) индексы перебираются в цикле очень удобно: от 0 включительно до длины списка не включительно. Таким образом можно перебрать все элементы списка.

Цикл for нередко используется и для формирования списка, если мы заранее знаем, сколько элементов в нём должно быть:

n = 10

a = []

**print**('Введите', n, 'значений:')

**for** i **in** range(n):

a.append(input())

**print**('Получился список строк:', a)

**6. Срезы списков**

Как и для строк, для списков определена операция взятия среза:

months = ['январь', 'февраль', 'март', 'апрель', 'май', 'июнь', 'июль',   
 'август', 'сентябрь', 'октябрь', 'ноябрь', 'декабрь']

spring = months[2:5] *# spring == ['март', 'апрель', 'май']*

**for** month **in** spring:

**print**(month)

**Важно**

Срезы можно использовать и для присваивания новых значений элементам списка. Например, если мы решим перевести на английский названия летних месяцев, то это можно сделать с помощью среза:

months[5:8] = ['June', 'July', 'August']

Теперь список months будет выглядеть так: [’январь’, ’февраль’, ’март’, ’апрель’, ’май’, ’June’, ’July’, ’August’, ’сентябрь’, ’октябрь’, ’ноябрь’, ’декабрь’]

**7. Списки и массивы**

Во многих языках программирования (да и в самом Python — в недрах стандартной библиотеки) имеется другой тип данных с похожими свойствами — массив. Поэтому списки иногда называют массивами, хоть это и не совсем правильно.

**Важно**

Элементы массива имеют одинаковый тип и располагаются в памяти одним куском, а элементы списка могут быть разбросаны по памяти как угодно и могут иметь разный тип. Всё это замедляет работу списков по сравнению с массивами, но придаёт им гораздо большую гибкость. Из этого различия вытекает и «Питонский путь» формирования списка: не создавать «пустой массив» и заполнять его значениями, а append’ить значения к изначально пустому списку.

[Справка](https://yandex.ru/support/lyceum-students)

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках проекта «Яндекс.Лицей», принадлежат АНО ДПО «ШАД». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «ШАД».

© 2018 – 2020  ООО «[Яндекс](https://yandex.ru/)»

Чаты