Тема урока: строки

- **Создание строки.** Для создания строк, мы используем парные кавычки '' или "":
- s1 = 'Python' s2 = "Pascal“
- **Считывание строки.** Для считывания текстовых данных в строковую переменную, мы используем функцию input():
- s = input() # считали текст num = int(input()) # считали текст и преобразовали его в целое число
- **Пустая строка.** Для создания пустой строки, мы пишем s = '' или s = "". Пустая строка – это аналог числа 0.

- **Длина строки.** Для определения длины строки (количества символов), мы используем встроенную функцию len():
- s = 'Hello'
- n = len(s) # значение переменной равно 5 print(n)

Конкатенация и умножение на число. Операторы + и * можно использовать для строк.
Оператор + сцепляет две и более строк. Это называется конкатенацией строк.
Оператор * повторяет строку указанное количество раз.

Выражение	Результат
'AB' + 'cd'	'ABcd'
'A' + '7' + 'B'	'A7B'
'Hi'* 4	'HiHiHiHi'

- **Оператор принадлежности in.** С помощью оператора in, мы можем проверять, находится ли одна строка в составе другой. То есть, является ли одна строка подстрокой другой:
- s = 'All you need is love'
- if 'love' in s:
- print('❤️')
- else: print('💔')

Индексация строк

- Очень часто бывает необходимо обратиться к конкретному символу в строке. Для этого в Python используются квадратные скобки [], в которых указывается индекс (номер) нужного символа в строке.
- В отличие от многих языков программирования, в Python есть возможность работы с отрицательными индексами. Если первый символ строки имеет индекс 0, то последнему элементу присваивается индекс -1.

Итерирование строк

- s = 'abcdef'
- for i in range(len(s)): print(s[i])
- Результатом выполнения такой программы будут строки:
- a
- b
- C
- d
- e
- f

- Если нам не нужен индекс самого символа, то мы можем использовать более короткий способ итерации:
- s = 'abcdef‘
- for c in s: print(c)

Срезы строк

При построении
среза s[x:y] первое число – это то место, где начинается срез (**включительно**), а второе – это место, где заканчивается срез (**невключительно**). Разрезая строки, мы создаем подстроку, которая по сути является строкой внутри другой строки.

Если опустить второй параметр в срезе s[x:] (но поставить двоеточие), то срез берется до конца строки. Аналогично если опустить первый параметр s[:y], то можно взять срез от начала строки. Срез s[:] совпадает с самой строкой s

Мы также можем использовать отрицательные индексы для создания срезов. Как уже говорилось ранее, отрицательные индексы строки начинаются с -1 и отсчитываются до достижения начала строки. При использовании отрицательных индексов ***первый параметр среза должен быть меньше второго, либо должен быть пропущен***.

Мы можем передать в срез третий необязательный параметр, который отвечает за шаг среза. К примеру, срез s[1:7:2] создаст строку bdf состоящую из каждого второго символа (индексы 1, 3, 5, правая граница не включена в срез).

- Если в качестве шага среза указать **отрицательное число**, то символы будут идти в обратном порядке.
- Следующий программный код:
- print(s[::-1])выводит:
- jihgfedcba

s = 'abcdefghij'

Программный код	Результат	Пояснение
s[2:5]	cde	строка состоящая из символов с индексами 2, 3, 4
s[:5]	abcde	первые пять символов строки
s[5:]	fghij	строка состоящая из символов с индексами от 5 до конца
s[-2:]	ij	последние два символа строки
s[:]	abcdefghij	вся строка целиком
s[1:7:2]	bdf	строка состоящая из каждого второго символа с индексами от 1 до 6
s[::-1]	jihgfedcba	строка в обратном порядке, так как шаг отрицательный

- Предположим, у нас есть строка s = 'abcdefghij' и мы хотим заменить символ с индексом 4 на 'X'. Можно попытаться написать код:
- s[4] = 'X‘
- Если мы хотим поменять какой-либо символ строки s, мы должны создать новую строку. Следующий код использует срезы и решает поставленную задачу:
- s = s[:4] + 'X' + s[5:]

Метод capitalize()

- Метод capitalize() возвращает копию строки s, в которой первый символ имеет верхний регистр, а все остальные символы имеют нижний регистр.
- Результатом выполнения следующего кода:

```
s = 'foO BaR BAZ quX'
print(s.capitalize())
```

Foo bar baz qux

s = 'foo123#BAR#.' print(s.capitalize()) Foo123#bar#

Метод swapcase()

- Метод swapcase() возвращает копию строки s, в которой все символы, имеющие верхний регистр, преобразуются в символы нижнего регистра и наоборот.
- Результатом выполнения следующего кода:
- s = 'FOO Bar 123 baz qUX' print(s.swapcase()) будет:
- foo bAR 123 BAZ Qux

Метод title()

- Метод title() возвращает копию строки s, в которой первый символ каждого слова переводится в верхний регистр.
- Результатом выполнения следующего кода:
- s = 'the sun also rises' print(s.title())будет:
- The Sun Also RisesЭтот метод использует довольно простой алгоритм: он не пытается различить важные и неважные слова и не обрабатывает аббревиатуры и апострофы. Результатом выполнения следующего кода:
- s = "what's happened to ted's IBM stock?" print(s.title()) будет:
- What'S Happened To Ted'S Ibm Stock

Метод lower()

- Метод lower() возвращает копию строки s, в которой все символы имеют нижний регистр.
- Результатом выполнения следующего кода:
- s = 'FOO Bar 123 baz qUX' print(s.lower()) будет:
- foo bar 123 baz qux

Метод upper()

- Метод upper() возвращает копию строки s, в которой все символы имеют верхний регистр.
- Результатом выполнения следующего кода:
- s = 'FOO Bar 123 baz qUX' print(s.upper()) будет:
- FOO BAR 123 BAZ QUX

Метод count()

- Метод count(<sub>, <start>, <end>) считает количество **непересекающихся** вхождений подстроки <sub> в исходную строку s.
- Результатом выполнения следующего кода:
- s = 'foo goo moo' print(s.count('oo')) print(s.count('oo', 0, 8)) # подсчет с 0 по 7 символбудет:
- 3
- 2

- На вход программе подается строка генетического кода, состоящая из букв А (аденин), Г (гуанин), Ц (цитозин), Т (тимин). Напишите программу, которая подсчитывает сколько аденина, гуанина, цитозина и тимина входит в данную строку генетического кода.
- **Формат входных данных**
  На вход программе подается строка генетического кода, состоящая из символов А, Г, Ц, Т, а, г, ц, т.
- **Формат выходных данных**
  Программа должна вывести сколько гуанина, тимина, цитозина, аденина входит в данную строку генетического кода.
- **Примечание.** Строка не содержит символов, кроме как А, Г, Ц, Т, а, г, ц, т.
- Тестовые данные 🟢
- **Sample Input 1:**
- АааГГЦЦцТТттт**Sample Output 1:**
- Аденин: 3 Гуанин: 2 Цитозин: 3 Тимин: 5**Sample Input 2:**
- ааггццттААГГЦЦТТ**Sample Output 2:**
- Аденин: 4 Гуанин: 4 Цитозин: 4 Тимин: 4

Метод startswith()

- Метод startswith(<suffix>, <start>, <end>) определяет **начинается** ли исходная строка s подстрокой <suffix>. Если исходная строка начинается с подстроки <suffix>, метод возвращает значение True, а если нет, то False.
- Результатом выполнения следующего кода:
- s = 'foobar' print(s.startswith('foo')) print(s.startswith('baz'))будет:
- True False

Метод endswith()

- Метод endswith(<suffix>, <start>, <end>) определяет **оканчивается** ли исходная строка s подстрокой <suffix>. Метод возвращает значение True если исходная строка оканчивается на подстроку <suffix> и False в противном случае.
- Результатом выполнения следующего кода:
- s = 'foobar' print(s.endswith('bar')) print(s.endswith('baz'))будет:
- True False

Методы find(), rfind()

- Метод find(<sub>, <start>, <end>) находит **индекс первого вхождения** подстроки <sub> в исходной строке s. Если строка s не содержит подстроки <sub>, то метод возвращает значение -1. Мы можем использовать данный метод наравне с оператором in для проверки: содержит ли заданная строка некоторую подстроку или нет.
- Результатом выполнения следующего кода:
- s = 'foo bar foo baz foo qux' print(s.find('foo')) print(s.find('bar')) print(s.find('qu')) print(s.find('python'))будет:
- 0 4 20 -1Метод rfind(<sub>, <start>, <end>) идентичен методу find(<sub>, <start>, <end>), за тем исключением, что он ищет первое вхождение подстроки <sub> начиная с конца строки s

Методы index(), rindex()

- Метод index(<sub>, <start>, <end>) идентичен методу find(<sub>, <start>, <end>), за тем исключением, что он **вызывает ошибку** ValueError: substring not found во время выполнения программы, если подстрока <sub> не найдена.
- Метод rindex(<sub>, <start>, <end>) идентичен методу index(<sub>, <start>, <end>), за тем исключением, что он ищет первое вхождение подстроки <sub> начиная с конца строки s.
- Методы find() и rfind() являются более безопасными чем index() и rindex(), так как не приводят к возникновению ошибки во время выполнения программы.

Метод strip()

- Метод strip() возвращает копию строки s у которой удалены все пробелы стоящие **в начале и конце** строки.
- Результатом выполнения следующего кода:
- s = ' foo bar foo baz foo qux ' print(s.strip()) будет:
- foo bar foo baz foo qux

Метод lstrip()

- Метод lstrip() возвращает копию строки s у которой удалены все пробелы стоящие **в начале** строки.
- Результатом выполнения следующего кода:
- s = ' foo bar foo baz foo qux ' print(s.lstrip()) будет:
- foo bar foo baz foo qux␣ ␣ ␣ ␣ ␣ ␣

Метод replace()

- Метод replace(<old>, <new>) возвращает копию s **со всеми** вхождениями подстроки <old>, замененными на <new>.
- Результатом выполнения следующего кода:
- s = 'foo bar foo baz foo qux' print(s.replace('foo', 'grault'))будет:
- grault bar grault baz grault quxМетод replace() может принимать опциональный третий аргумент <count>, который определяет количество замен.
- Результатом выполнения следующего кода:
- s = 'foo bar foo baz foo qux' print(s.replace('foo', 'grault', 2)) будет:
- grault bar grault baz foo qux

Метод rstrip()

- Метод rstrip() возвращает копию строки s у которой удалены все пробелы стоящие **в конце** строки.
- Результатом выполнения следующего кода:
- s = ' foo bar foo baz foo qux ' print(s.rstrip()) будет:
- ␣ ␣ ␣ ␣ ␣ ␣foo bar foo baz foo qux

Метод isalnum()

- Метод isalnum() определяет, состоит ли исходная строка из буквенно-цифровых символов. Метод возвращает значение True если исходная строка является непустой и состоит **только** из буквенно-цифровых символов и False в противном случае.
- Результатом выполнения следующего кода:
- s1 = 'abc123' s2 = 'abc$*123' s3 = '' print(s1.isalnum()) print(s2.isalnum()) print(s3.isalnum())будет:
- True False False

Метод isalpha()

- Метод isalpha() определяет, состоит ли исходная строка из буквенных символов. Метод возвращает значение True если исходная строка является непустой и состоит **только** из буквенных символов и False в противном случае.
- Результатом выполнения следующего кода:
- s1 = 'ABCabc' s2 = 'abc123' s3 = '' print(s1.isalpha()) print(s2.isalpha()) print(s3.isalpha())будет:
- True False False

Метод isdigit()

- Метод isdigit() определяет, состоит ли исходная строка **только** из цифровых символов. Метод возвращает значение True если исходная строка является непустой и состоит **только** из цифровых символов и False в противном случае.
- Результатом выполнения следующего кода:
- s1 = '1234567' s2 = 'abc123' s3 = '' print(s1.isdigit()) print(s2.isdigit()) print(s3.isdigit())будет:
- True False False

Метод islower()

- Метод islower() определяет, являются ли **все** буквенные символы исходной строки строчными (имеют нижний регистр). Метод возвращает значение True если все буквенные символы исходной строки являются строчными и False в противном случае. **Все неалфавитные символы игнорируются!**
- Результатом выполнения следующего кода:
- s1 = 'abc' s2 = 'abc1$d' s3 = 'Abc1$D' print(s1.islower()) print(s2.islower()) print(s3.islower()) будет:
- True True False

Метод isupper()

- Метод isupper() определяет, являются ли **все** буквенные символы исходной строки заглавными (имеют верхний регистр). Метод возвращает значение True если все буквенные символы исходной строки являются заглавными и False в противном случае. **Все неалфавитные символы игнорируются!**
- Результатом выполнения следующего кода:
- s1 = 'ABC' s2 = 'ABC1$D' s3 = 'Abc1$D' print(s1.isupper()) print(s2.isupper()) print(s3.isupper())будет:
- True True False

Метод isspace()

- Метод isspace() определяет, состоит ли исходная строка **только** из пробельных символов. Метод возвращает значение True если строка состоит только из пробельных символов и False в противном случае.
- Результатом выполнения следующего кода:
- s1 = ' ' s2 = 'abc1$d' print(s1.isspace()) print(s2.isspace())будет:
- True False

Форматирование строк

- Хранить строки в переменных удобно, но часто бывает необходимо **собирать строки** из других объектов (строк, чисел и т.д.) и выполнять с ними нужные манипуляции. Для этой цели можно воспользоваться механизмом **форматирования строк**.
- Рассмотрим следующий код:
- age = 27 txt = 'My name is Timur, I am ' + age print(txt)Такой код приводит к ошибке во время выполнения программы, поскольку мы пытаемся сложить число и строку. Для решения такой проблемы мы можем использовать функцию str, которая преобразует числовое значение в строку:
- age = 27 txt = 'My name is Timur, I am ' + str(age) print(txt)Такой код работает, однако в Python предпочтительным способом форматирования считается использование метода format. Предыдущую программу можно переписать в виде:
- age = 27 txt = 'My name is Timur, I am {}'.format(age) print(txt)Мы передаем необходимые параметры методу format, а Python форматирует указанную строку и помещает их в строку на место заполнителей {}. Мы можем создавать сколько угодно заполнителей в строке:
- age = 27 name = 'Timur' profession = 'math teacher' txt = 'My name is {}, I am {}, I work as a {}'.format(name, age, profession) print(txt)Для наглядности и гибкости форматирования мы можем использовать порядковый номер в заполнителе: {0}, {1}, {2},.... Такой номер определяет позицию параметра, переданного методу format (нумерация начинается с нуля):
- age = 27 name = 'Timur' profession = 'math teacher' txt = 'My name is {0}, I am {1}, I work as a {2}'.format(name, age, profession) print(txt)Параметр name встает в {0} заполнитель, параметр age встает в {1} заполнитель и т. д. Мы можем использовать одно и тоже число в нескольких заполнителях
- name = 'Timur' txt = 'My name is {0}-{0}-{0}'.format(name) print(txt)Результатом выполнения такого кода будет:
- My name is Timur-Timur-Timur

f-строки

- Метод format хорошо справляется с задачей форматирования строк, однако если параметров много, то код может показаться немного избыточным:
- first_name = 'Timur' last_name = 'Guev' age = 27 profession = 'math teacher' affiliation = 'BeeGeek' print('Hello, {0} {1}. You are {2}. You are a {3}. You were a member of {4}' .format(first_name, last_name, age, profession, affiliation))В Python 3.6 появилась новая разновидность строк — так называемые f-строки. Если поставить перед строкой префикс f, в заполнители можно будет включить код, например имя переменной. Предыдущий код можно записать в виде:
- first_name = 'Timur' last_name = 'Guev' age = 27 profession = 'math teacher' affiliation = 'BeeGeek' print(f'Hello, {first_name} {last_name}. You are {age}. You are a {profession}. You were a member of {affiliation}')На место заполнителя {first_name} встает значение переменной first_name, на место заполнителя {last_name} встает значение переменной last_name и т.д.

Функция ord

- Функция ord позволяет определить код некоторого символа в таблице символов Unicode. Аргументом данной функции является одиночный символ.
- Результатом выполнения следующего кода:
- num1 = ord('A') num2 = ord('B') num3 = ord('a') print(num1, num2, num3)будет:
- 65 66 97Обратите внимание, что функция ord принимает именно **одиночный символ**. Если попытаться передать строку, содержащую более одного символа:
- num = ord('Abc') print(num)мы получим ошибку времени выполнения:
- TypeError: ord() expected a character, but string of length 3 found

Функция chr

- Функция chr позволяет определить по коду символа сам символ. Аргументом данной функции является численный код.
- Результатом выполнения следующего кода:
- chr1 = chr(65) chr2 = chr(75) chr3 = chr(110) print(chr1, chr2, chr3)будет:
- A K n

- Функции ord и chr часто работают в паре. Мы можем использовать следующий код для вывода всех заглавных букв английского алфавита:
- for i in range(26): print(chr(ord('A') + i))Вызов функции ord('A') возвращает код символа «A», который равен 65. Далее на каждой итерации цикла, к данному коду прибавляется значение переменной i = 0, 1, 2, ..., 25, а затем полученный код преобразуется в символ с помощью вызова функции chr

Списки

- **Создание списка**
- Чтобы создать список, нужно перечислить его элементы через запятую в квадратных скобках:
- numbers = [2, 4, 6, 8, 10] languages = ['Python', 'C#', 'C++', 'Java']
  Список numbers состоит из 5 элементов, и каждый из них — целое число.
- numbers[0] == 2;
- numbers[1] == 4;
- numbers[2] == 6;
- numbers[3] == 8;
- numbers[4] == 10.
- Список languages состоит из 4 элементов, каждый из которых — **строка.**
- languages[0] == 'Python';
- languages[1] == 'C#';
- languages[2] == 'C++';
- languages[3] == 'Java'.

Пустой список

- Создать пустой список можно двумя способами:
- Использовать пустые квадратные скобки [];
- Использовать встроенную функцию, которая называется list.
- Следующие две строки кода создают пустой список:
- mylist = [] # пустой список mylist = list() # пустой список

Вывод списка

- Для вывода всего списка можно применить функцию print():
- numbers = [2, 4, 6, 8, 10] languages = ['Python', 'C#', 'C++', 'Java'] print(numbers) print(languages) Функция print() выводит на экран элементы списка, в квадратных скобках, разделенные запятыми:
- [2, 4, 6, 8, 10] ['Python', 'C#', 'C++', 'Java']Обратите внимание, что вывод списка содержит квадратные скобки. Позже мы научимся выводить элементы списка в более удобном виде с помощью циклов.

Встроенная функция list

- Python имеет встроенную функцию list(), которая помимо создания пустого списка может преобразовывать некоторые типы объектов в списки.
- Например, мы знаем, что функция range() создает последовательность целых чисел в заданном диапазоне. Для преобразования этой последовательности в список, мы пишем следующий код:
- numbers = list(range(5))Во время исполнения этого кода происходит следующее:
- Вызывается функция range(), в которую в качестве аргумента передается число 5;
- Эта функция возвращает последовательность чисел 0, 1, 2, 3, 4;
- Последовательность чисел 0, 1, 2, 3, 4 передается в качестве аргумента в функцию list();
- Функция list() возвращает список [0, 1, 2, 3, 4];
- Список [0, 1, 2, 3, 4] присваивается переменной numbers.
- Вот еще один пример:
- even_numbers = list(range(0, 10, 2)) # список содержит четные числа 0, 2, 4, 6, 8 odd_numbers = list(range(1, 10, 2)) # список содержит нечетные числа 1, 3, 5, 7, 9Точно также с помощью функции list() мы можем создать список из символов строки. Для преобразования строки в список мы пишем следующий код:
- s = 'abcde' chars = list(s) # список содержит символы 'a', 'b', 'c', 'd', 'e'Во время исполнения этого кода происходит следующее:
- Вызывается функция list(), в которую в качестве аргумента передается строка 'abcde';
- Функция list() возвращает список ['a', 'b', 'c', 'd', 'e'];
- Список ['a', 'b', 'c', 'd', 'e'] присваивается переменной chars.

Функция len()

- **Длиной списка** называется количество его элементов. Для того, чтобы посчитать длину списка мы используем встроенную функцию len() (от слова length – длина).
- Следующий программный код:
- numbers = [2, 4, 6, 8, 10] languages = ['Python', 'C#', 'C++', 'Java'] print(len(numbers)) # выводим длину списка numbers print(len(languages)) # выводим длину списка languages print(len(['apple', 'banana', 'cherry'])) # выводим длину списка, состоящего из 3 элементоввыведет:
- 5 4 3

Оператор принадлежности in

- Оператор in позволяет проверить, содержит ли список некоторый элемент.
- Рассмотрим следующий код:
- numbers = [2, 4, 6, 8, 10] if 2 in numbers: print('Список numbers содержит число 2') else: print('Список numbers не содержит число 2')Такой код проверяет, содержит ли список numbers число 2 и выводит соответствующий текст:
- Список numbers содержит число 2Мы можем использовать оператор in вместе с логическим оператором not. Например
- numbers = [2, 4, 6, 8, 10] if 0 not in numbers: print('Список numbers не содержит нулей')

Срезы

- Рассмотрим список numbers = [2, 4, 6, 8, 10].
- С помощью среза мы можем получить несколько элементов списка, создав диапазон индексов разделенных двоеточием numbers[x:y].
- Следующий программный код:
- print(numbers[1:3]) print(numbers[2:5])выводит:
- [4, 6] [6, 8, 10]При построении среза numbers[x:y] первое число – это то место, где начинается срез (**включительно**), а второе – это место, где заканчивается срез (**невключительно**). Разрезая списки, мы создаем новые списки, по сути, подсписки исходного.
- При использовании срезов со списками мы также можем опускать второй параметр в срезе numbers[x:] (но поставить двоеточие), тогда срез берется до конца списка. Аналогично если опустить первый параметр numbers[:y], то можно взять срез от начала списка.
- Срез numbers[:] возвращает копию исходного списка.
- Как и в строках, мы можем использовать отрицательные индексы в срезах списков.

Операция конкатенации + и умножения на число *

- Мы можем применять операторы + и * для списков подобно тому как мы это делали со строками.
- Следующий программный код:
- print([1, 2, 3, 4] + [5, 6, 7, 8]) print([7, 8] * 3) print([0] * 10)выводит:
- [1, 2, 3, 4, 5, 6, 7, 8] [7, 8, 7, 8, 7, 8] [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Отличие списков от строк

- Несмотря на всю схожесть списков и строк, есть одно очень важное отличие: строки — **неизменяемые** объекты, а списки – **изменяемые**.
- Следующий программный код:
- s = 'abcdefg' s[1] = 'x' # пытаемся изменить 2 символ (по индексу 1) строки приводит к ошибке:
- object does not support item assignmentСледующий программный код:
- numbers = [1, 2, 3, 4, 5, 6, 7] numbers[1] = 101 # изменяем 2 элемент (по индексу 1) списка print(numbers)выводит:
- [1, 101, 3, 4, 5, 6, 7]

Метод append()

- Для добавления нового элемента **в конец списка** используется метод append().
- Следующий программный код:
- numbers = [1, 1, 2, 3, 5, 8, 13] # создаем список numbers.append(21) # добавляем число 21 в конец списка numbers.append(34) # добавляем число 34 в конец списка print(numbers)выведет:
- [1, 1, 2, 3, 5, 8, 13, 21, 34]Обратите внимание, для того чтобы использовать метод append(), нужно, чтобы список был создан, при этом он может быть пустым.
- Следующий программный код:
- numbers = [] # создаем пустой список numbers.append(1) numbers.append(2) numbers.append(3) print(numbers)выведет:
- [1, 2, 3]

Метод extend()

- Можно также расширить список другим списком, путем вызова метода extend().
- Следующий программный код:
- numbers = [0, 2, 4, 6, 8, 10] odds = [1, 3, 5, 7] numbers.extend(odds) print(numbers) выведет:
- [0, 2, 4, 6, 8, 10, 1, 3, 5, 7]Метод extend() как бы расширяет один список, добавляя к нему элементы другого списка.
- Отличие между методами append() и extend() проявляется при добавлении строки к списку.
- Следующий программный код:
- words1 = ['iq option', 'stepik', 'beegeek'] words2 = ['iq option', 'stepik', 'beegeek'] words1.append('python') words2.extend('python') print(words1) print(words2)выведет:
- ['iq option', 'stepik', 'beegeek', 'python'] ['iq option', 'stepik', 'beegeek', 'p', 'y', 't', 'h', 'o', 'n'] Метод append() добавляет строку 'python' целиком к списку, а метод extend() разбивает строку 'python' на символы 'p', 'y', 't', 'h', 'o', 'n' и их добавляет в качестве элементов списка.

.

Удаление элементов

- С помощью оператора del можно удалять элементы списка по определенному индексу.
- Следующий программный код:
- numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9] del numbers[5] # удаляем элемент имеющий индекс 5 print(numbers)выведет:
- [1, 2, 3, 4, 5, 7, 8, 9]Элемент под указанным индексом удаляется, а список перестраивается.

- Оператор del работает и со срезами: мы можем удалить целый диапазон элементов списка.
- Следующий программный код:
- numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9] del numbers[2:7] # удаляем элементы с 2 по 6 включительно print(numbers)выведет:
- [1, 2, 8, 9]Мы можем удалить все элементы на четных позициях (0, 2, 4, ...) исходного списка.
- Следующий программный код:
- numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9] del numbers[::2] print(numbers) выведет:
- [2, 4, 6, 8]

Вывод с помощью цикла for

- Для вывода элементов списка **каждого на отдельной строке** можно использовать следующий код:
- **Вариант 1.** Если нужны индексы элементов:
- numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] for i in range(len(numbers)): print(numbers[i])Мы передаем в функцию range() длину списка len(numbers). В нашем случае длина списка numbers, равна 11. Таким образом вызов функции range(len(numbers)) имеет вид range(11) и переменная цикла i последовательно перебирает все значения от 0 до 10. Это означает, что выражение numbers[i] последовательно вернет все элементы списка numbers. Такой способ итерации списка удобен, когда нам нужен не только сам элемент numbers[i], но и его индекс i.
- **Вариант 2.** Если индексы не нужны:
- numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] for num in numbers: print(num)Этот цикл пройдет по списку numbers, придавая переменной цикла num значение каждого элемента списка (!) в отличии от предыдущего цикла, в котором переменная цикла «бегала» по индексам списка.
- Если требуется выводить элементы списка на одной строке, через пробел, то мы можем использовать необязательный параметр end функции print():
- numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] for num in numbers: print(num, end=' ')

Вывод с помощью распаковки списка

- **Вариант 1.** Вывод элементов списка через один символ пробела:
- numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] print(*numbers)Такой код выведет:
- 0 1 2 3 4 5 6 7 8 9 10

- вывод элементов списка, каждого на отдельной строке
- numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] print(*numbers, sep='\n')Такой код выведет:
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

- Поскольку строки содержат символы, подобно тому, как списки содержат элементы, то мы можем использовать распаковку строк точно так же, как и распаковку списков.
- Такой код:
- s = 'Python' print(*s)
- print()
- print(*s, sep='\n')выведет:
- P
- Y
- t
- h
- o
- n
- P y t h o n

Метод split()

- Метод split() разбивает строку на слова, используя в качестве разделителя последовательность пробельных символов.
- Следующий программный код:
- s = 'Python is the most powerful language' words = s.split() print(words)выведет:
- ['Python', 'is', 'the', 'most', 'powerful', 'language']

Метод join()

- Метод join() собирает строку из элементов списка, используя в качестве разделителя строку, к которой применяется метод.
- Следующий программный код:
- words = ['Python', 'is', 'the', 'most', 'powerful', 'language'] s = ' '.join(words) print(s) выведет:
- Python is the most powerful language

- Существует большая разница между результатами вызова методов s.split() и s.split(' '). Разница в поведении проявляется когда строка содержит несколько пробелов между словами.
- Следующий программный код:
- s = 'Python is the most powerful language' words1 = s.split() words2 = s.split(' ') print(words1) print(words2) выведет списки:
- ['Python', 'is', 'the', 'most', 'powerful', 'language'] ['Python', '', '', '', 'is', '', '', 'the', '', 'most', '', 'powerful', '', 'language']**Примечание 2.** Методы split() и join() являются строковыми методами. Следующий код приводит к ошибке:
- print([1, 2].split()) print([1, 2].join([3, 4, 5])) **Примечание 3.** Строковый метод join() работает только со списком строк. Следующий код приводит к ошибке:
- numbers = [1, 2, 3, 4] # список чисел s = '*'.join(numbers) print(s)

Метод insert()

- Метод insert() позволяет вставлять значение в список в заданной позиции. В него передается два аргумента:
- index: индекс, задающий место вставки значения;
- value: значение, которое требуется вставить.
- Когда значение вставляется в список, список расширяется в размере, чтобы разместить новое значение. Значение, которое ранее находилось в заданной индексной позиции, и все элементы после него сдвигаются на одну позицию к концу списка.
- Следующий программный код:
- names = ['Gvido', 'Roman' , 'Timur'] print(names) names.insert(0, 'Anders') print(names) names.insert(3, 'Josef') print(names)выведет:
- ['Gvido', 'Roman' , 'Timur'] ['Anders', 'Gvido', 'Roman' , 'Timur'] ['Anders', 'Gvido', 'Roman' , 'Josef', 'Timur']Если указан недопустимый индекс, то во время выполнения программы не происходит ошибки. Если задан индекс за пределами конца списка, то значение будет добавлено в конец списка. Если применен отрицательный индекс, который указывает на недопустимую позицию, то значение будет вставлено в начало списка.

Метод index()

- Метод index() возвращает индекс первого элемента, значение которого равняется переданному в метод значению. Таким образом, в метод передается один параметр:
- value: значение, индекс которого требуется найти.
- Если элемент в списке не найден, то во время выполнения происходит ошибка.
- Следующий программный код:
- names = ['Gvido', 'Roman' , 'Timur'] position = names.index('Timur') print(position)выведет:
- 2

Метод remove()

- Метод remove() удаляет первый элемент, значение которого равняется переданному в метод значению. В метод передается один параметр:
- value: значение, которое требуется удалить.
- Метод уменьшает размер списка на один элемент. Все элементы после удаленного элемента смещаются на одну позицию к началу списка. Если элемент в списке не найден, то во время выполнения происходит ошибка.
- Следующий программный код:
- food = ['Рис', 'Курица', 'Рыба', 'Брокколи', 'Рис'] print(food) food.remove('Рис') print(food)выведет:
- ['Рис', 'Курица', 'Рыба', 'Брокколи', 'Рис'] ['Курица', 'Рыба', 'Брокколи', 'Рис']

Метод pop()

- Метод pop() удаляет элемент по указанному индексу и возвращает его. В метод pop() передается один **необязательный** аргумент:
- index: индекс элемента, который требуется удалить.
- Если индекс не указан, то метод удаляет и возвращает последний элемент списка. Если список пуст или указан индекс за пределами диапазона, то во время выполнения происходит ошибка.
- Следующий программный код:
- names = ['Gvido', 'Roman' , 'Timur'] item = names.pop(1) print(item) print(names)выведет:
- Roman ['Gvido', 'Timur']

Метод count()

- Метод count() возвращает количество элементов в списке, значения которых равны переданному в метод значению.
- Таким образом, в метод передается один параметр:
- value: значение, количество элементов, равных которому, нужно посчитать.
- Если значение в списке не найдено, то метод возвращает 0.
- Следующий программный код:
- names = ['Timur', 'Gvido', 'Roman', 'Timur', 'Anders', 'Timur'] cnt1 = names.count('Timur') cnt2 = names.count('Gvido') cnt3 = names.count('Josef') print(cnt1) print(cnt2) print(cnt3)выведет:
- 3 1 0

Метод reverse()

- Метод reverse() инвертирует порядок следования значений в списке, то есть меняет его на противоположный.
- Следующий программный код:
- names = ['Gvido', 'Roman' , 'Timur'] names.reverse() print(names)

- выведет:
- ['Timur', 'Roman', 'Gvido']

Метод clear()

- Метод clear() удаляет все элементы из списка.
- Следующий программный код:
- names = ['Gvido', 'Roman' , 'Timur'] names.clear() print(names)выведет:
- []

Метод copy()

- Метод copy() создает поверхностную копию списка.
- Следующий программный код:
- names = ['Gvido', 'Roman' , 'Timur'] names_copy = names.copy() # создаем поверхностную копию списка names print(names) print(names_copy)выведет:
- ['Gvido', 'Roman', 'Timur'] ['Gvido', 'Roman', 'Timur'] Аналогичного результата можно достичь с помощью срезов или функции list():
- names = ['Gvido', 'Roman' , 'Timur'] names_copy1 = list(names) # создаем поверхностную копию с помощью функции list() names_copy2 = names[:] # создаем поверхностную копию с помощью среза от начала до конца

Метод sort()

- В Python списки имеют встроенный метод sort(), который сортирует элементы списка по возрастанию или убыванию.
- Следующий программный код:
- a = [1, 7, -3, 9, 0, -67, 34, 12, 45, 1000, 6, 8, -2, 99] a.sort() print('Отсортированный список:', a)выведет:
- Отсортированный список: [-67, -3, -2, 0, 1, 6, 7, 8, 9, 12, 34, 45, 99, 1000]По умолчанию метод sort() сортирует список по возрастанию. Если требуется отсортировать список по убыванию, необходимо явно указать параметр reverse = True.
- Следующий программный код:
- a = [1, 7, -3, 9, 0, -67, 34, 12, 45, 1000, 6, 8, -2, 99] a.sort(reverse = True) # сортируем по убыванию print('Отсортированный список:', a)выведет:
- Отсортированный список: [1000, 99, 45, 34, 12, 9, 8, 7, 6, 1, 0, -2, -3, -67]