Урок 6.  
Списки. Работа со строками.

короткая линия

# План урока

1. Списки
2. Методы split() и join()
3. Срезы
4. Форматирование строк

# Списки

Знакомясь с кортежами, мы говорили о том, что они представляют собой неизменяемые контейнеры различных типов данных. Сегодня мы поговорим о мощном инструменте – типе данных под названием **список.** Списки также используются для хранения данных, но, в отличие от кортежей, они изменяемые. Это значит, что вы можете добавлять, удалять, а также заменять и сортировать элементы списка. Пустой список объявляется квадратными скобками.

a = []

Также вы можете инициализировать список с предустановленными значениями:

even\_numbers = [2, 4, 6, 8, 10]

Существует несколько способов расширить наш список. Рассмотрим некоторые из них:

1. Добавить элемент в конец списка. К списку применяется метод append(), в скобках указывается добавляемый элемент.

even\_numbers.append(12)

Примечание: инструкция append() отличается от привычных уже функций. append() “прикрепляется” к объекту (в данном случае к списку) с помощью точки. Команды такого вида называются **методами**. Подробнее об этом мы будем говорить позже.

2) Расширение списка.

even\_numbers + [12, 14, 16, 18, 20]

Более предпочтительный вариант реализации с помощью метода extend():

even\_numbers.extend([12, 14, 16, 18, 20])

3) Получение списка, в котором n фиксированных элементов.

|  |
| --- |
| ones = [1] \* 100  *# получаем список из ста единиц* |

К определенному элементу списка можно обратиться также, как и в кортежах – по числовому индексу. В очередной раз заметим, что индексация начинается с нуля. Однако, обратившись к элементу списка по индексу, можно не только посмотреть его, но и заменить на другой элемент.

|  |
| --- |
| powers\_of\_ten = [10, 100, 1000] powers\_of\_ten.append(1) *# ошибка - единица не является степенью десятки* powers\_of\_ten[-1] = 10 \*\* 4 *# исправляем ошибку* |

Попробуйте выполнить замену символа в строке похожим образом. Что произойдет?

**Интересно:**

1. К определенному символу строки можно также обратиться по индексу. Например:

|  |
| --- |
| s = **"Hello!"**  print(s[1], s[-2]) *# выведет сначала "e", затем "o"* |

2) Так же, как строку можно перевести в кортеж с помощью функции tuple(), ее можно преобразовать в список, используя функцию list():

|  |
| --- |
| s = **"Any string"**  print(list(s)) *# выведет ["A", "n", "y", " ", "s", "t", "r", "i", "n", "g"]* |

Стоит заметить, что элементами списка могут быть объекты любого типа данных, к тому же не обязательно, чтобы элементы были только одного типа. Вы можете создавать двумерные списки (“список в списке”), трехмерные и даже n-мерные.

**Задача “Сохраните четные”**

**Задача “Медиана”**

**Задача “Обработка списка”**

**Задача “Помедленнее, я записываю!”**

**Задача “Шпаргалка политика”**

**Задача “Мальчик, который кричал “Питон””**

# split() и join()

Часто, когда строка и список используются в связке, применяются методы (вспомним, что с одним методом – append() – мы уже познакомились) split() и join().

Метод split() применяется к строкам и “разбивает” (от слова split) данную строку на список подстрок. В круглых скобках можно передать подстроку, по которой разбивается строка (если ничего не передавать, строка разбивается по пробелам). В то время как append() изменяет список, split() возвращает значение – список (также как функция max(), например, возвращает число).

|  |
| --- |
| s = **"This is a string with spaces"** s = s.split() *# s = ["This", "is", "a", "string", "with", "spaces"]* |

Метод join() является его противоположностью – он возвращает строку, “соединенную” (или “склеенную”) из списка строк через какую-то заданную строку. Допустим, что у нас есть список строк s, взятый из предыдущего примера. Изменим строку:

|  |
| --- |
| s = **"!"**.join(s) *# s = "This!is!a!string!with!spaces"* |

Мы “склеили” подстроки с помощью символа “!”.

**Задача “Логин для регистрации”**

**Задача “Разбить IP”**

**Задача “Ликбез наоборот”**

# Срезы

Срезы – уникальный и очень гибкий инструмент языка Python, позволяющий брать некоторую часть списка или строки. Вот что можно сделать с помощью срезов: взять каждый второй (третий, четвертый, n-ый) элемент списка/символ строки, отбросить любое число элементов/символов с конца или с начала и даже развернуть строку или список.

Используются они на удивление просто и интуитивно. Рассмотрим примеры:

|  |
| --- |
| s = **"Lorem ipsum dolor sit amet, consectetur adipiscing elit"** print(s[3]) *# так мы обращаемся к четвертому элементу* print(s[1:]) *# так мы отрезаем нулевой элемент строки (т. е. печатаем строку с первого элемента)* print(s[2:10]) *# так мы вырезаем из строки подстроку со 2 по 9 символ включительно* print(s[3:11:2]) *# так мы вырезаем из строки подстроку с 3 по 9 символ включительно с шагом 2* print(s[12:1:-1]) *# так мы вырезаем из строки подстроку с 12 по 2 символ включительно (идем в обратном порядке)* |

Использование срезов в списках аналогично приведенным выше примерам.

В общем виде их использование можно представить так: s[start:stop:step]. Если присмотреться, похожая концепция наблюдается в цикле for (собственно, можно обойтись без срезов, используя циклы, но это гораздо менее комфортный способ).

**Задача “Агрессивный маркетинг”**

**Задача “Примитивный шифр”**

# Форматирование строк

Нельзя не упомянуть про гибкий и полезный инструмент форматирования строк. Он намного удобнее обычной конкатенации и используется в большинстве случаев. Метод format() позволяет вставлять в заранее подготовленную строку нужные Вам переменные в нужном порядке. Вы пишете обычную строку в кавычках, однако вместо подставляемых значений вставляете фигурные скобки. К этой строке применяется метод format(); в круглых скобках указываются аргументы, которые следует подставить вместо фигурных скобок. Решим задачу “Любимая книга” (<https://stepik.org/lesson/142208/step/8?unit=116414>) с помощью format():

|  |
| --- |
| author, book = input(), input()  print(**"{} — автор бестселлера {} — выпустил новую книгу! Спешите приобрести ее в магазине Питонист!"**.format(author, book)) |

Возможны и другие сценарии работы с format. Например, можно изменить порядок следования аргументов.

|  |
| --- |
| first, second, third = input(), input(), input()  *# обычный порядок, можно было не писать индексы в фигурных скобках*  print(**"Рецепт: сначала добавим {0}, потом {1}, а в конце — {2}"**.format(first, second, third))  *# измененный порядок следования*  print(**"Рецепт: сначала добавим {2}, потом {0}, а в конце — {1}"**.format(first, second, third)) |
|  |

Полный список сценариев использования можно найти в документации:

<https://docs.python.org/3/library/string.html#format-examples>

**Домашнее задание:**

**Задача “Квадраты в обратном порядке”**

**Задача “Грамотное логирование”**

**Задача “Ответ от сервера”**

**Задача “Восстановить выражение”**