Строки. Индексация. Срезы.

На занятии мы углубим свои знания о строках. Теперь мы сможем не только считывать строку, но и работать с ней, в том числе делать посимвольный перебор. Познакомимся с новым методом извлечения подстроки — срезами.

Строка как коллекция

На прошлом занятии мы познакомились с коллекцией, которая называется **множество**. Вспомним, что основная особенность коллекций — возможность хранить несколько значений под одним именем. Можно сказать, что коллекция является **контейнером** для этих значений.

Но еще до изучения множеств мы уже знали тип данных, который ведет себя подобно коллекции. Этот тип данных — строка. Действительно, ведь строка фактически является последовательностью символов. В некоторых языках программирования есть специальный тип данных **char**, позволяющий хранить один символ. В Python такого типа данных нет, поэтому можно сказать, что строка — **последовательность односимвольных строк**.

Что мы знаем о строках

Давайте вспомним, что мы уже знаем о работе со строками в языке Python. Мы умеем создавать строки четырьмя способами: задавать напрямую, считывать с клавиатуры функцией input(), преобразовывать число в строку функцией str и склеивать из двух других строк операцией +. Кроме того, мы умеем узнавать длину строки, используя функцию len, и проверять, является ли одна строка частью другой, используя операцию in:

```
fixed_word = 'опять'
print(fixed_word)
word = input()
print(word)
number = 25
number_string = str(number)
print(number_string)
word_plus_number = fixed_word + number_string
print(word_plus_number)
print(len(word_plus_number))
print('оп' in word_plus_number)
```

Индексация в строках

В отличие от множеств, в строках важен порядок элементов (символов). Действительно, если множества {1, 2, 3} и {3, 2, 1} — это одинаковые множества, то строки МИР и РИМ — две совершенно разные строки. Наличие порядка дает нам возможность пронумеровать символы. Нумерация символов начинается с 0.

Индекс

По индексу можно получить соответствующий ему символ строки. Для этого нужно после самой строки написать в квадратных скобках индекс символа.

```
word = 'привет'
initial_letter = word[0]
print(initial_letter)  # сделает то же, что print('п')
```

```
other_letter = word[3]
print(other_letter)  # сделает то же, что print('в')
```

Естественно, в этом примере word с тем же успехом можно было считать с клавиатуры через input(). Тогда мы не могли бы заранее сказать, чему равны переменные initial_letter и other_letter.

А что будет, если попытаться получить букву, номер которой слишком велик? В этом случае Python выдаст ошибку:

```
word = 'привет'
print(word[6]) # builtins.IndexError: string index out of
range
```

Конечно, номер в квадратных скобках — не всегда фиксированное число, которое прописано в самой программе. Его тоже можно считать с клавиатуры или получить в результате арифметического действия.

```
word = 'привет'
number_of_letter = int(input())  # предположим,
пользователь ввел 3
print(word[number_of_letter])  # тогда будет выведена
буква 'в'
```

Отрицательные индексы

Кроме «прямой» индексации (начинающейся с 0), в Python разрешены отрицательные индексы: word[-1] означает последний символ строки word, word[-2] — предпоследний и т д.

А можно ли, используя индексацию, изменить какой-либо символ строки? Давайте проверим:

```
word = 'карова'  # Написали слово с ошибкой
word[1] = 'о'    # Пробуем исправить, но:
# TypeError: 'str' object does not support item assignment
```

Важно! Интерпретатор Python выдает ошибку — значит, изменить отдельный символ строки невозможно, т. е. строка относится к **неизменяемым** типам данных в Python.

Перебор элементов строки

В предыдущем уроке мы узнали, что цикл for можно использовать для перебора элементов множества. Таким же образом можно использовать цикл for, чтобы перебрать все буквы в слове:

```
word = 'карова'  # Написали слово с ошибкой
word[1] = 'о'    # Пробуем исправить, но:
# TypeError: 'str' object does not support item assignment
```

Но, так как символы в строке пронумерованы, у нас есть еще один способ перебрать все элементы в строке: перебрать все индексы, используя уже знакомую нам конструкцию for i in range(...).

```
text = 'hello, my dear friends!'
vowels = 0
for letter in text:
    if letter in {'a', 'e', 'i', 'o', 'u', 'y'}:
        vowels += 1
print(vowels)
```

Хранение текстов в памяти компьютера

Давайте немного поговорим о том, как строки хранятся в памяти компьютера.

Кодирование

Поскольку компьютер умеет хранить только двоичные числа, для записи нечисловой информации (текстов, изображений, видео, документов) прибегают к кодированию.

Самый простой случай кодирования — сопоставление кодов текстовым символам.

Один самых распространенных форматов такого кодирования — таблица ASCII (American standard code for information interchange).

32 -	48 - 0	64 - @	80 - P	96 - '	112 - p
33 - !	49 - 1	65 - A	81 - Q	97 - a	113 - q
34 - "	50 - 2	66 - B	82 - R	98 - b	114 - r
35 - #	51- 3	67 - C	83 - S	99 - c	115 - s
36 - $	52 - 4	68 - D	84 - T	100 - d	116 - t
37 - %	53 - 5	69 - E	85 - U	101 - e	117 - u
38 -&	54 - 6	70 - F	86 - V	102 - f	118 - v
39 - '	55 - 7	71 - G	87 - W	103 - g	119 - w
40 - (	56 - 8	72 - H	88 - X	104 - h	120 - x
41 -)	57 - 9	73 - I	89 - Y	105 - i	121 - y
42 - *	58 - :	74 - J	90 - Z	106 - j	122 - z
43 - +	59 - ;	75 - K	91 - [	107 - k	123 - {
44 - ,	60 - <	76 - L	92 - \	108 - l	124 - |
45 - -	61 - =	77 - M	93 - j	109 - m	125 - }
46 - .	62 ->	78 - N	94 - ^	110 - n	126 - ~
47 - /	63 - ?	79 - O	95 - ÷	111 - o	127 - ⌂
48 -0	64 - @	80 - P	96 -	112 - p	

Изначально в этой таблице каждому символу был поставлен в соответствие 7-битный код, что позволяло идентифицировать 128 различных символов. В таблице вы не видите символы с кодами, меньшими 32, так как они являются служебными и не предназначены для непосредственного вывода на экран (пробел, перевод строки, табуляция и т. д.).

Этого хватало на латинские буквы обоих регистров, знаки препинания и спецсимволы — например, перевод строки или разрыв страницы. Позже код расширили до 1 байта, что позволяло хранить уже 256 различных значений: в таблицу помещались буквы второго алфавита (например, кириллица) и дополнительные графические элементы (псевдографика).

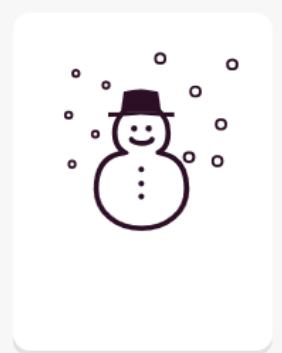
В некоторых относительно низкоуровневых языках (например, в C) можно в любой момент перейти от представления строки в памяти к последовательности байтов, начинающейся по какому-либо адресу.

Сейчас однобайтные кодировки отошли на второй план, уступив место Юникоду.

Юникод

Юникод — таблица, которая содержит соответствия между числом и каким-либо знаком, причем количество знаков может быть любым. Это позволяет одновременно использовать любые символы любых алфавитов и дополнительные графические элементы. Кроме того, в Юникоде каждый символ, помимо кода, имеет некоторые свойства: например, буква это или цифра. Это позволяет более гибко работать с текстами.

В Юникод все время добавляются новые элементы, а сам размер этой таблицы не ограничен и будет только расти, поэтому сейчас при хранении в памяти одного юникод-символа может потребоваться от 1 до 8 байт. Отсутствие ограничений привело к тому, что стали появляться символы на все случаи жизни. Например, есть несколько снеговиков.



Этого вы можете увидеть, если наберете:

```
print('\u2603')
```

Важно понять, что все строки в Python хранятся именно как последовательность юникод-символов.

Функция ord

Для того чтобы узнать код некоторого символа, существует функция ord (от order — «порядок»).

```
ord('Б')
```

```
1041
```

Функция chr

Зная код, всегда можно получить соответствующий ему символ. Для этого существует функция chr (от character — «символ»):

```
chr(1041)
```

```
'Б'
```

Функции ord и chr часто работают в паре. Попробуйте, например, предположить, что будет выведено на экран в результате работы следующей программы:

```
for i in range(26):

    print(chr(ord('A') + i))
```

Пример 1. Игра в города: один раунд

При игре в города игроки по очереди называют названия городов (или, возможно, любые другие слова) так, чтобы первая буква каждого нового слова совпадала с последней буквой предыдущего.

Напишите программу, которая считывает подряд две строки, после чего выводит «ВЕРНО», если последний символ первой строки совпадает с первым символом второй, и «НЕВЕРНО» в противном случае.

Формат ввода

Два слова — каждое в своей строке.

Формат вывода

Одно сообщение — «ВЕРНО» или «НЕВЕРНО».

Пример

Ввод	Вывод
париж житомир	ВЕРНО

```
w1 = input()
w2 = input()
if w1[-1] == w2[0]:
    print("ВЕРНО")
else:
    print("НЕВЕРНО")
```

Пример 2. Игра в города

Пользователь (или несколько пользователей за одним компьютером) вводит слова. Начиная со второго введённого слова, программа проверяет, совпадает ли первая буква свеже введённого слова с последней буквой предыдущего. Если да, то программа работает дальше (считывает очередное слово). Если нет — выводит последнее на этот момент введённое слово и завершает работу.

Формат ввода

Вводятся несколько строк подряд.

Формат вывода

Выводится одна строка.

Пример

Ввод	Вывод
новгород дублин новгород дублин тула	тула

Примечания: В данной задаче гарантируется, что будет введено как минимум два слова.

```
w1 = input()
x = int(input())
k = len(w1)
if x <= k:
    print(w1[x-1])
else:
    print("ОШИБКА")
```

Пример 3. Какая-то там буква

Напишите программу, которая считывает сообщение, затем номер. После этого программа выводит букву из сообщения с таким номером, причём считается, что номера букв отсчитываются с единицы.

Если введённое число не является правильным номером буквы, вывести «ОШИБКА».

Формат ввода

В первой строке записано сообщение, во второй — номер буквы.

Формат вывода

Одна буква или сообщение «ОШИБКА».

Пример 1

Ввод	Вывод
привет 2	р

Пример 2

Ввод	Вывод
привет -100	ОШИБКА

```
wl = input()
x = int(input())
k = len(wl)
if (x <= k) and (x >= 1):
    print(wl[x - 1])
else:
    print("ОШИБКА")
```

Пример 4. Цезарь его знает

Как известно, Цезарь тоже пользовался шифрованием сообщений, причем у него был свой способ. Сначала выбирается шаг шифрования (число), а затем все буквы послания заменяются на буквы, отстоящие от них в алфавите на шаг шифрования. Например, при шаге шифрования 3 (таким чаще всего пользовался Цезарь), буква А заменяется на букву Г, буква Б – на букву Д.

Обратите внимание, что алфавит «зациклен», то есть при сдвиге буквы Я на шаг 3 получится буква В.

Напишите программу, которая будет зашифровывать послание с помощью шифра Цезаря с заданным шагом шифрования.

Формат ввода

Две строки. Первая содержит шаг шифрования, вторая – послание.

Формат вывода

Строка с зашифрованным посланием.

Пример 1

Ввод	Вывод
3 АБВ	ГДЕ

Пример 2

Ввод	Вывод
5 На дворе трава, на траве дрова!	Те йзухк чхезе, те чхезк йхузе!

Примечания: Символы русского алфавита расположены в стандартной для Python таблице кодировки подряд, то есть номера, выдаваемые функцией ord(symbol), идут подряд. Буква «ё» идёт в таблице кодировки отдельно от основного алфавита. При решении задачи считайте, что буквы «ё» в русском алфавите нет.

```
x = int(input())
s = input()
for i in range(len(s)):
    if((ord(s[i]) >= 32 and ord(s[i]) <= 63)) or (ord(s[i]) >= 8211):
        print(s[i], end="")
    elif (ord(s[i]) + x) > 1103 and (ord(s[i])) < 8211:
        print(chr(ord(s[i]) + x - 32), end="")
    else:
        print(chr(ord(s[i]) + x), end="")

```

Пример 5. Ползём вниз

Изобразите извилистый спуск улитки по стене по заданной траектории (см. примеры).

Формат ввода

Вводится одна строка.

Начальный символ этой строки — символ рисования пути улитки.

Далее следует последовательность символов “<”, “>” и “V”, которые означают движение улитки, соответственно, влево, вправо и вниз.

Гарантируется, что по любой горизонтали движение происходит только в одну сторону (иными словами, между идущими в любом порядке “<” и “>” всегда есть хотя бы один “V”) и что путь не пройдёт левее начального положения.

Формат вывода

Выводятся несколько строк, состоящих из символов рисования пути и пробелов. Верхний левый угол соответствует начальному положению улитки. Каждый последующий символ входной строки означает, что нужно дорисовать символ рисования пути левее, правее или ниже предыдущего. Пробелы используются для отступов.

Пример 1

Ввод	Вывод
.>>>VV<<V	....

Пример 2

Ввод	Вывод

@VVV>>>>>>>V<<VVVV<<<VV>>>	@
	@
	@
	@@@@@@@@@
	  @@@
	  @
	  @
	  @
	@@@@
	@
	@@@@

```
s = input()
char = s[0]
k = s[1:].split('V')
b = 0
for i in k:
    if i and i[0] == '<':
        b -= len(i)
    print(' ' * b + char * (1 + len(i)))
    if i and i[0] == '>':
        b += len(i)
```

Пример 6. Счастливый билетик

Рассмотрим еще одну задачу. Билет называют счастливым по-питерски, если сумма цифр его номера, стоящих на четных местах, равна сумме цифр, стоящих на нечетных местах. Нам необходимо написать программу, которая определяет, является ли билет счастливым по-питерски.

Если рассматривать номер билета как строку из цифр, задача сводится к подсчету суммы цифр, стоящих на позициях 0, 2, 4..., и суммы цифр, стоящих на позициях 1, 3, 5... Чтобы перебрать элементы, мы можем воспользоваться конструкцией for i in range(...), указав шаг 2. Тогда соответствующий фрагмент программы может выглядеть следующим образом:

```
number = input()
odd = even = 0
for i in range(0, len(number), 2):
    odd += int(number[i])
for i in range(1, len(number), 2):
    even += int(number[i])
if odd == even:
    print('Счастливый по-питерски!')
```

Подумайте, как можно решить данную задачу за один цикл.

Срезы строк

На примере разобранной задачи мы увидели, что перебор элементов строки с помощью конструкции for i in range(...) является достаточно гибким: можно перебрать не все индексы, можно идти с шагом, скажем, 2 или даже −1, то есть в обратном порядке. Но существует способ без всякого цикла преобразовать строку нужным образом: взять отдельный ее кусок, символы с нечетными номерами и т. д. Этот способ — **срез (slice)**.

Срез строки

В самом простом варианте срез строки — ее кусок от одного индекса включительно и до другого не включительно (как для range). То есть это новая, более короткая строка.

Срез записывается с помощью квадратных скобок, в которых указывается начальный и конечный индекс, разделенные двоеточием.

```
text = 'Hello, world!'
print(text[0:5])
print(text[7:12])
```

Если не указан **начальный индекс**, срез берется от начала (от 0). Если не указан **конечный индекс**, срез берется до конца строки. Попробуйте предположить, что будет выведено на экран, если в предыдущей программе записать срезы следующим образом:

```
text = 'Hello, world!'
print(text[:5])
print(text[7:])
```

Разрешены отрицательные индексы для отсчета с конца списка. В следующем примере из строки, содержащей фамилию, имя и отчество, будет извлекаться фамилия.

```
full_name = 'Иванов И. И.'
surname = full_name[:-6]
```

Как и для range, в параметры среза можно добавить третье число — **шаг обхода**. Этот параметр не является обязательным и записывается через второе двоеточие. Вот как может выглядеть программа «Счастливый билет», если решать ее с помощью срезов:

```
number = input()
odd = even = 0

# срез будет от начала строки до конца с шагом два: 0, 2, 4,...
for n in number[::2]:
    odd += int(n)

# срез от второго элемента строки до конца с шагом два: 1, 3, 5,...
for n in number[1::2]:
    even += int(n)

if odd == even:
    print('Счастливый по-питерски!')
```

Интересное отличие среза от обращения по индексу к отдельному элементу состоит в том, что мы не получим ошибку при указании границ среза за пределами строки. В срез в таком случае попадут только те элементы, которые находятся по валидным индексам среза:

```
a = 'Python'
print(a[2:10000]) # thon
print(a[999:]) # пустая строка
```

Шаг может быть и отрицательным — для прохода по строке в обратном порядке. Если в этом случае не указать начальный и конечный индекс среза, ими станут последний и первый индексы строки соответственно (а не наоборот, как при положительном шаге):

```
text = 'СЕЛ В ОЗЕРЕ БЕРЕЗОВ ЛЕС'
text_reversed = text[::-1]
print(text == text_reversed)
```

Итак, с помощью квадратных скобок можно получить доступ как к одному символу строки, так и к некоторой последовательности символов, причем совсем необязательно идущих подряд!

Пример 1. Быки и коровы

Напишите программу, обрабатывающую один раунд игры «Быки и коровы». Пользователь вводит две строки. Гарантируется, что это две строки одинаковой длины и что все символы в каждой из них разные. Необходимо вывести отдельно количество быков — символов, которые есть в обеих

строках и стоят на одном и том же месте, и количество коров — символов, которые есть в обеих строках, но на разных местах.

Формат ввода

Две строки.

Формат вывода

Два целых числа через пробел — количество быков и коров.

Пример

Ввод	Вывод
питон пилот	3 1

```
x = input()
y = input()
k1 = 0
k2 = 0
for i in range(len(x)):
    for j in range(len(y)):
        if x[i] == y[j] and i == j:
            k1 = k1 + 1
        if x[i] == y[j] and i != j:
            k2 = k2 + 1
print(k1, k2)

```

Пример 2. Шах и мат, программисты

Напишите программу, которая выводит обозначения клеток шахматной доски. Клетки нумеруются (заглавными) латинскими буквами слева направо и натуральными числами снизу вверх, после каждого обозначения клетки следует пробел. Доска квадратная, размер вводится с клавиатуры и не превышает 9.

Формат ввода

Натуральное число, не превышающее 9 — размер доски.

Формат вывода

Шахматная доска в формате, описанном в условии и показанном в примере.

Пример

Ввод	Вывод
4	A4 B4 C4 D4 A3 B3 C3 D3 A2 B2 C2 D2 A1 B1 C1 D1

```
x = int(input())
for i in range(x, 0, -1):
    for j in range(65, 65 + x):
        print(chr(j), i, " ", sep='', end="")
    print()
```

Пример 3. Розенкранц и Гильденстерн меняют профессию

Ограничение времени	1 секунда
Ограничение памяти	64Mb
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Второстепенные герои пьесы Шекспира «Гамлет» Розенкранц и Гильденстерн появляются и в пьесе Тома Стоппарда. Они подбрасывают монетку, и Гильденстерна интересует, какое максимальное количество орлов подряд может выпасть. (Розенкранца это не интересует.)

Вводится одна строка, каждая буква которой представляет собой результат одного броска монетки — «о» обозначает орла, «р» обозначает решку. Программа должна вывести максимальное количество орлов, выпавших подряд.

Формат ввода

Одна строка, состоящая из букв «о» и «р» — результаты бросков.

Формат вывода

Одно целое число — максимальное число орлов, выпавших подряд.

Пример

Ввод	Вывод
poorpooop	3

```
x = input()
max = 0
x1 = x[0:].split('p')
for i in x1:
    if len(i) > max:
        max = len(i)
print(max)

```

Пример 4. Фильтр

Напишите программу, которая проводит первичную обработку неких сложных и глючных логов. Нужно удалить сочетание «%%» в начале некоторых строк и удалить строки, начинающиеся с «####».

Формат ввода

На первой строке вводится натуральное число N — количество строк, подлежащих обработке.

Далее вводятся сами строки, N штук.

Формат вывода

Нужно вывести те же строки в том же порядке, однако если строка начинается с символов «%%», то их выводить не следует, а если строка начинается с сочетания символов «####», то её нужно вообще пропустить.

Пример

Ввод	Вывод
3 SVO TRS 29481292 %%LJPZ DME 11113283675 ####&%^^^^	SVO TRS 29481292 LJPZ DME 11113283675

```
x = int(input())
for i in range(x):
    y = input()
    if y[0:2] == "%%":
        print(y[2:])
    elif y[0:4] != "####":
        print(y)


```

Методы

Метод — это функция, применяемая к объекту, в данном случае — к строке. Метод вызывается в виде Имя_объекта.Имя_метода(параметры). Например, S.find("e") — это применение к строке S метода find с одним параметром "e".

Методы find и rfind

Метод find находит в данной строке (к которой применяется метод) данную подстроку (которая передается в качестве параметра). Функция возвращает индекс первого вхождения искомой подстроки. Если же подстрока не найдена, то метод возвращает значение -1.

```
S = 'Hello'
print(S.find('e'))
# вернёт 1
print(S.find('ll'))
# вернёт 2
print(S.find('L'))
# вернёт -1
```

Аналогично, метод rfind возвращает индекс последнего вхождения данной строки (“поиск справа”).

```
S = 'Hello'
print(S.find('l'))
# вернёт 2
print(S.rfind('l'))
# вернёт 3
```

Если вызвать метод find с тремя параметрами S.find(T, a, b), то поиск будет осуществляться в срезе S[a:b]. Если указать только два параметра S.find(T, a), то поиск будет осуществляться в срезе S[a:], то есть начиная с символа с индексом a и до конца строки. Метод S.find(T, a, b) возращает индекс в строке S, а не индекс относительно среза.

Метод replace

Метод replace заменяет все вхождения одной строки на другую. Формат: S.replace(old, new) — заменить в строке S все вхождения подстроки old на подстроку new. Пример:

```
print('Hello'.replace('l', 'L'))
# вернёт 'HeLLo'
```

Если методу replace задать еще один параметр: S.replace(old, new, count), то заменены будут не все вхождения, а только не больше, чем первые count из них.

```
print('Abrakadabra'.replace('a', 'A', 2))
# вернёт 'AbrAkAdabra'
```

Метод count

Подсчитывает количество вхождений одной строки в другую строку. Простейшая форма вызова S.count(T) возвращает число вхождений строки T внутри строки S. При этом подсчитываются только непересекающиеся вхождения, например:

```
print('Abracadabra'.count('a'))
# вернёт 4
```

```
print(('a' * 10).count('aa'))
# вернёт 5
```

При указании трех параметров S.count(T, a, b), будет выполнен подсчет числа вхождений строки T в срезе S[a:b].

Задания на лабораторную работу

(для выполнения заданий необходимо посмотреть так же материал лабораторной работы 4)

Задание 1. Напишите программу, которая последовательно получает на вход имя, отчество, фамилию и должность сотрудника, а затем преобразует имя и отчество в инициалы, добавляя должность после запятой.

Пример ввода:

Алексей
Константинович
Романов
бухгалтер

Вывод:

А. К. Романов, бухгалтер

Задание 2. Напишите программу для подсчета количества пробелов и непробельных символов в введенной пользователем строке.

Пример ввода:

В роще, травы шевеля, мы нащиплем щавеля

Вывод:

Количество пробелов: 6, количество других символов: 34

Задание 3. Дана строка. Замените в этой строке все появления буквы h на букву H, кроме первого и последнего вхождения.

Задание 4. Дана строка, состоящая из слов, разделенных пробелами. Определите, сколько в ней слов. Используйте для решения задачи метод count.

Задание 5. Дана строка, состоящая ровно из двух слов, разделенных пробелом. Переставьте эти слова местами. Результат запишите в строку и выведите получившуюся строку.

Списки

*На занятии рассматривается новый тип данных — списки (*list*).*

Список - это непрерывная динамическая коллекция элементов. Каждому элементу списка присваивается порядковый номер - его индекс. Первый индекс равен нулю, второй - единице и так далее. Основные операции для работы со списками - это индексирование, срезы, добавление и удаление элементов, а также проверка на наличие элемента в последовательности.

Создание пустого списка выглядит так:

```
empty_list = []
```

Создадим список, состоящий из нескольких чисел:

```
numbers = [40, 20, 90, 11, 5]
```

Настало время строковых переменных:

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
```

Не будем забывать и о дробях:

```
fractions = [3.14, 2.72, 1.41, 1.73, 17.9]
```

Мы можем создать список, состоящий из различных типов данных:

```
values = [3.14, 10, 'Hello world!', False, 'Python is the best']
```

И такое возможно

```
list_of_lists = [[2, 4, 0], [11, 2, 10], [0, 19, 27]]
```

Индексирование

Индексирование обозначает операцию обращения к элементу по его порядковому номеру напоминаем, что нумерация начинается с нуля. Проиллюстрируем это на примере:

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
print(fruits[0])
print(fruits[1])
print(fruits[4])

>>> Apple
>>> Grape
>>> Orange
```

Списки в Python являются изменяемым типом данных. Мы можем изменять содержимое каждой из ячеек:

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
fruits[0] = 'Watermelon'
```

```
fruits[3] = 'Lemon'
print(fruits)

>>> ['Watermelon', 'Grape', 'Peach', 'Lemon', 'Orange']
```

Индексирование работает и в обратную сторону. Как такое возможно? Всё просто, мы обращаемся к элементу списка по отрицательному индексу. Индекс с номером -1 дает нам доступ к последнему элементу, -2 к предпоследнему и так далее.

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
print(fruits[-1])
print(fruits[-2])
print(fruits[-3])
print(fruits[-4])

>>> Orange
>>> Banan
>>> Peach
>>> Grape
```

Создание списка с помощью list()

Переходим к способам создания списка. Самый простой из них был приведен выше. Еще раз для закрепления:

```
smiles = ['(ಠ_ಠ)', '(□ ¯)', '( °  °)', '(□°-°)□']
```

Создание списка с помощью функции list() В неё мы можем передать любой итерируемый объект.

Рассмотрим несколько примеров:

```
letters = list('abcdef')
numbers = list(range(10))
even_numbers = list(range(0, 10, 2))
print(letters)
print(numbers)
print(even_numbers)

>>> ['a', 'b', 'c', 'd', 'e', 'f'
>>> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [0, 2, 4, 6, 8]
```

Длина списка

Как узнать длину списка? Можно, конечно, просто посчитать количество элементов. Есть функция len() – возвращает длину любой итерируемой переменной, переменной, по которой можно запустить цикл. Рассмотрим пример:

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
print(len(fruits))

>>> 5
```

```
numbers = [40, 20, 90]
print(len(numbers))

>>> 3
```

"...любой итерируемой", а это значит:

```
string = 'Hello world'
print(len(string))
# 11

>>> 11

print(len(range(10))

>>> 10
```

Срезы

В начале занятий мы говорили о "срезах". Давайте вспомнимм подробнее, что это такое. Срезом называется некоторая подпоследовательность. Принцип действия срезов очень прост: мы "отрезаем" кусок от исходной последовательности элемента, не меняя её при этом. Я сказал "последовательность", а не "список", потому что срезы работают и с другими итерируемыми типами данных, например, со строками.

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
part_of_fruits = fruits[0:3]
print(part_of_fruits)

>>> ['Apple', 'Grape', 'Peach']
```

Детально рассмотрим синтаксис срезов:

```
итерируемая_переменная[начальный_индекс:конечный_индекс - 1:длина_шага]
```

Обращае ваше внимание, что мы делаем срез от начального индекса до конечного индекса - 1. То есть i = начальный_индекс и i < конечный индекс

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
print(fruits[0:1])
# Если начальный индекс равен 0, то его можно опустить
print(fruits[:2])
print(fruits[:3])
print(fruits[:4])
print(fruits[:5])
# Если конечный индекс равен длине списка, то его тоже можно опустить
print(fruits[:len(fruits)])
print(fruits[::])

>>> ['Apple']
>>> ['Apple', 'Grape']
>>> ['Apple', 'Grape', 'Peach']
>>> ['Apple', 'Grape', 'Peach', 'Banan']
>>> ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
```

```
>>> ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
>>> ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
```

Самое время понять, что делает третий параметр среза - длина шага!

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
print(fruits[::2])
print(fruits[::3])
# Длина шага тоже может быть отрицательной!
print(fruits[::-1])
print(fruits[4:2:-1])
print(fruits[3:1:-1])

>>> ['Apple', 'Peach', 'Orange']
>>> ['Apple', 'Banan']
>>> ['Orange', 'Banan', 'Peach', 'Grape', 'Apple']
>>> ['Orange', 'Banan']
>>> ['Banan', 'Peach']
```

А теперь вспоминаем всё, что мы знаем о циклах. В Python их целых два! Цикл for и цикл while Нас интересует цикл for, с его помощью мы можем перебирать значения и индексы наших последовательностей. Начнем с перебора значений:

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
for fruit in fruits:
    print(fruit, end=' ')

>>> Apple Grape Peach Banan Orange
```

В переменную fruit объявленную в цикле по очереди записываются значения всех элементов списка fruits

```
for index in range(len(fruits)):
    print(fruits[index], end=' ')
```

В данном примере разберемся, что делает функция range(len(fruits))

Мы с вами знаем, что функция len() возвращает длину списка, а range() генерирует диапазон целых чисел от 0 до len()-1.

Сложив 2+2, мы получим, что переменная index принимает значения в диапазоне от 0 до len()-1. Идем дальше, fruits[index] - это обращение по индексу к элементу с индексом index списка fruits. А так как переменная index принимает значения всех индексов списка fruits, то в цикле мы переберем значения всех элементов нашего списка!

Операция in

С помощью in мы можем проверить наличие элемента в списке, строке и любой другой итерируемой переменной.

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
if 'Apple' in fruits:
    print('В списке есть элемент Apple')
```

```
>>> В списке есть элемент Apple

fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
if 'Lemon' in fruits:
   print('В списке есть элемент Lemon')
else:'
   print('В списке НЕТ элемента Lemon')

>>> В списке НЕТ элемента Lemon
```

Приведу более сложный пример:

```
all_fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
my_favorite_fruits = ['Apple', 'Banan', 'Orange']
for item in all_fruits:
   if item in my_favorite_fruits:
      print(item + ' is my favorite fruit')
   else:
      print('I do not like ' + item)

>>> Apple is my favorite fruit
>>> I do not like Grape
>>> I do not like Peach
>>> Banan is my favorite fruit
>>> Orange is my favorite fruit
```

Методы для работы со списками

Начнем с метода append(), который добавляет элемент в конец списка:

```
# Создаем список, состоящий из четных чисел от 0 до 8 включительно
numbers = list(range(0,10,2))
# Добавляем число 200 в конец списка
numbers.append(200)
numbers.append(1)
numbers.append(2)
numbers.append(3)
print(numbers)

>>> [0, 2, 4, 6, 8, 200, 1, 2, 3]
```

Мы можем передавать методу append() абсолютно любые значения:

```
all_types = [10, 3.14, 'Python', ['I', 'am', 'list']]
all_types.append(1024)
all_types.append('Hello world!')
all_types.append([1, 2, 3])
print(all_types)

>>> [10, 3.14, 'Python', ['I', 'am', 'list'], 1024, 'Hello world!', [1, 2, 3]]
```

Метод append() отлично выполняет свою функцию. Но, что делать, если нам нужно добавить элемент в середину списка? Это умеет метод insert(). Он добавляет элемент в список на произ-

вольную позицию. insert() принимает в качестве первого аргумента позицию, на которую нужно вставить элемент, а вторым — сам элемент.

```
# Создадим список чисел от 0 до 9
numbers = list(range(10))
# Добавление элемента 999 на позицию с индексом 0
numbers.insert(0, 999)
print(numbers) # первый print
numbers.insert(2, 1024)
print(numbers) # второй print
numbers.insert(5, 'Засланная строка-шпион')
print(numbers) # третий print

>>> [999, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9] # первый print
>>> [999, 0, 1024, 1, 2, 3, 4, 5, 6, 7, 8, 9] # второй print
>>> [999, 0, 1024, 1, 2, 'Засланная строка-шпион', 3, 4, 5, 6, 7, 8, 9] # третий print
```

Добавлять элементы в список мы научились, осталось понять, как их из него удалять. Метод pop() удаляет элемент из списка по его индексу:

```
numbers = list(range(10))
print(numbers) # 1
# Удаляем первый элемент
numbers.pop(0)
print(numbers) # 2
numbers.pop(0)
print(numbers) # 3
numbers.pop(2)
print(numbers) # 4
# Чтобы удалить последний элемент, вызовем метод pop без аргументов
numbers.pop()
print(numbers) # 5
numbers.pop()
print(numbers) # 6

>>> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] # 1
>>> [1, 2, 3, 4, 5, 6, 7, 8, 9] # 2
>>> [2, 3, 4, 5, 6, 7, 8, 9] # 3
>>> [2, 3, 5, 6, 7, 8, 9] # 4
>>> [2, 3, 5, 6, 7, 8] # 5
>>> [2, 3, 5, 6, 7] # 6
```

Теперь мы знаем, как удалять элемент из списка по его индексу. Но что, если мы не знаем индекса элемента, но знаем его значение? Для такого случая у нас есть метод remove(), который удаляет первый найденный по значению элемент в списке.

```
all_types = [10, 'Python', 10, 3.14, 'Python', ['I', 'am', 'list']]
all_types.remove(3.14)
print(all_types) # 1
all_types.remove(10)
print(all_types) # 2
all_types.remove('Python')
print(all_types) # 3
```

```
>>> [10, 'Python', 10, 'Python', ['I', 'am', 'list']] # 1
>>> ['Python', 10, 'Python', ['I', 'am', 'list']] # 2
>>> [10, 'Python', ['I', 'am', 'list']] # 3
```

А сейчас немного посчитаем, посчитаем элементы списка с помощью метода count()

```
numbers = [100, 100, 100, 200, 200, 500, 500, 500, 500, 500, 999]
print(numbers.count(100)) # 1
print(numbers.count(200)) # 2
print(numbers.count(500)) # 3
print(numbers.count(999)) # 4

>>> 3 # 1
>>> 2 # 2
>>> 5 # 3
>>> 1 # 4
```

В программировании, как и в жизни, проще работать с упорядоченными данными, в них легче ориентироваться и что-либо искать. Метод sort() сортирует список по возрастанию значений его элементов.

```
numbers = [100, 2, 11, 9, 3, 1024, 567, 78]
numbers.sort()
print(numbers) # 1
fruits = ['Orange', 'Grape', 'Peach', 'Banan', 'Apple']
fruits.sort()
print(fruits) # 2

>>> [2, 3, 9, 11, 78, 100, 567, 1024] # 1
>>> ['Apple', 'Banan', 'Grape', 'Orange', 'Peach'] # 2
```

Мы можем изменять порядок сортировки с помощью параметра reverse. По умолчанию этот параметр равен False

```
fruits = ['Orange', 'Grape', 'Peach', 'Banan', 'Apple']
fruits.sort()
print(fruits) # 1
fruits.sort(reverse=True)
print(fruits) # 2

>>> ['Apple', 'Banan', 'Grape', 'Orange', 'Peach'] # 1
>>> ['Peach', 'Orange', 'Grape', 'Banan', 'Apple'] # 2
```

Иногда нам нужно перевернуть список для этого есть метод reverse():

```
numbers = [100, 2, 11, 9, 3, 1024, 567, 78]
numbers.reverse()
print(numbers) # 1
fruits = ['Orange', 'Grape', 'Peach', 'Banan', 'Apple']
fruits.reverse()
print(fruits) # 2

>>> [78, 567, 1024, 3, 9, 11, 2, 100] # 1
>>> ['Apple', 'Banan', 'Peach', 'Grape', 'Orange'] # 2
```

Допустим, у нас есть два списка и нам нужно их объединить. Для этого есть полезный метод extend(). Этот метод вызывается для одного списка, а в качестве аргумента ему передается другой список, extend() записывает в конец первого из них начало второго:

```
fruits = ['Banana', 'Apple', 'Grape']
vegetables = ['Tomato', 'Cucumber', 'Potato', 'Carrot']
fruits.extend(vegetables)
print(fruits)

>>> ['Banana', 'Apple', 'Grape', 'Tomato', 'Cucumber', 'Potato', 'Carrot']
```

Существует специальный метод для очистки списка — clear()

```
fruits = ['Banana', 'Apple', 'Grape']
vegetables = ['Tomato', 'Cucumber', 'Potato', 'Carrot']
fruits.clear()
vegetables.clear()
print(fruits)
print(vegetables)

>>> []
>>> []
```

Метод index() возвращает индекс элемента. Работает это так: вы передаете в качестве аргумента в index() значение элемента, а метод возвращает его индекс:

```
fruits = ['Banana', 'Apple', 'Grape']
print(fruits.index('Apple'))
print(fruits.index('Banana'))
print(fruits.index('Grape'))

>>> 1
>>> 0
>>> 2
```

Метод copy(), копирует список и возвращает его брата-близнеца.

Во-первых, если мы просто присвоим уже существующий список новой переменной, то на первый взгляд всё выглядит неплохо:

```
fruits = ['Banana', 'Apple', 'Grape']
new_fruits = fruits
print(fruits)
print(new_fruits)

>>> ['Banana', 'Apple', 'Grape']
>>> ['Banana', 'Apple', 'Grape']
```

Но есть одно маленькое "НО":

```
fruits = ['Banana', 'Apple', 'Grape']
new_fruits = fruits
fruits.pop()
```

```
print(fruits)
print(new_fruits)

# Внезапно, из списка new_fruits исчез последний элемент
>>> ['Banana', 'Apple']
>>> ['Banana', 'Apple']
```

При прямом присваивании списков копирования не происходит. Обе переменные начинают ссылаться на один и тот же список! То есть если мы изменим один из них, то изменится и другой. Что же тогда делать? Пользоваться методом copy(), конечно:

```
fruits = ['Banana', 'Apple', 'Grape']
new_fruits = fruits.copy()
fruits.pop()
print(fruits)
print(new_fruits)

>>> ['Banana', 'Apple']
>>> ['Banana', 'Apple', 'Grape']
```

Но что если у нас список в списке? Скопируется ли внутренний список с помощью метода copy() — нет:

```
fruits = ['Banana', 'Apple', 'Grape', ['Orange','Peach']]
new_fruits = fruits.copy()
fruits[-1].pop()
print(fruits) # 1
print(new_fruits) # 2

>>> ['Banana', 'Apple', 'Grape', ['Orange']] # 1
>>> ['Banana', 'Apple', 'Grape', ['Orange']] # 2
```

Рассмотри еще примеры.

Пример 1.

Дан список некоторых целых чисел, найдите значение 20 в нем и, если оно присутствует, замените его на 200. Обновите список только при первом вхождении числа 20.

Мы можем использовать метод index(), который позволит получить индекс первого вхождения некоторого объекта (в нашем случае числа 20). Затем просто изменим элемент списка с этим индексом до нужного нам значения (то есть 200).

```
list1 = [5, 10, 15, 20, 25, 50, 20]

index = list1.index(20)

list1[index] = 200

print(list1)
```

Пример 2

Дан список некоторых целых чисел. Необходимо вывести список в обратном порядке.

Самым простым решением станет срез. При указании шага среза -1, мы получим тот же список, но в обратном порядке.

```
aList = [100, 200, 300, 400, 500]

aList = aList[::-1]

print(aList)
```

Задания на практическую работу

1. Создайте список из 10 четных чисел и выведите его с помощью цикла for

2. Создайте список из 5 элементов. Сделайте срез от второго индекса до четвертого

3. Создайте пустой список и добавьте в него 10 случайных чисел и выведите их. В данной задаче нужно использовать функцию randint.

```
from random import randint
n = randint(1, 10) # Случайное число от 1 до 10
```

4.Удалите все элементы из списка, созданного в задании 3

5.Создайте список из введенной пользователем строки и удалите из него символы 'a', 'e', 'o'

6.Даны два списка, удалите все элементы первого списка из второго

```
a = [1, 3, 4, 5]
b = [4, 5, 6, 7]
# Вывод
>>> [6, 7]
```

7.Создайте список из случайных чисел и найдите наибольший элемент в нем.

8.Найдите наименьший элемент в списке из задания 7

9.Найдите сумму элементов списка из задания 7

10.Найдите среднее арифметическое элементов списка из задания 7

Сложные задачи

1.Создайте список из случайных чисел. Найдите номер его последнего локального максимума (локальный максимум — это элемент, который больше любого из своих соседей).

2.Создайте список из случайных чисел. Найдите максимальное количество его одинаковых элементов.

3.Создайте список из случайных чисел. Найдите второй максимум.

```
a = [1, 2, 3] # Первый максимум == 3, второй == 2
```

4.Создайте список из случайных чисел. Найдите количество различных элементов в нем.