11.2 Основы работы со списками

Шаг 1

Тема урока: основы работы со списками

1. Встроенные функции `len()`, `sum()`, `min()`, `max()`
2. Оператор принадлежности `in`
3. Индексация и срезы
4. Конкатенация и умножение на число
5. Отличие списков от строк

Аннотация. Начинаем работать со списками.

Основы работы со списками

Работа со списками очень сильно напоминает работу со строками, поскольку и списки, и строки содержат отдельные элементы. Однако элементы списка могут иметь произвольный тип, а элементами строк всегда являются символы. Многое из того, что мы делали со строками, доступно и при работе со списками.

Функция len()

Длиной списка называется количество его элементов. Для того чтобы посчитать длину списка, мы используем встроенную функцию `len()` (от слова length – длина).

Приведённый ниже 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 элементов
```

выводит:

```
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 2
```

Мы 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')
```

Индексация

При работе со строками мы использовали **индексацию**, то есть обращение к конкретному символу строки по его индексу. Аналогично, можно индексировать и списки.

Для индексации списков в Python используются квадратные скобки `[]`, в которых указывается индекс (номер) нужного элемента в списке:

Пусть `numbers = [2, 4, 6, 8, 10]`.

Таблица ниже показывает, как работает индексация:

Выражение	Результат	Пояснение
`numbers[0]`	2	первый элемент списка
`numbers[1]`	4	второй элемент списка
`numbers[2]`	6	третий элемент списка
`numbers[3]`	8	четвертый элемент списка
`numbers[4]`	10	пятый элемент списка

Обратите внимание: первый элемент списка `numbers[0]`, а не `numbers[1]`. Программисты же всё считают с нуля.

Так же, как и в строках, для нумерации с конца разрешены отрицательные индексы.

Выражение	Результат	Пояснение
`numbers[-1]`	10	пятый элемент списка
`numbers[-2]`	8	четвертый элемент списка
`numbers[-3]`	6	третий элемент списка
`numbers[-4]`	4	второй элемент списка
`numbers[-5]`	2	первый элемент списка

Как и в строках, попытка обратиться к элементу списка по несуществующему индексу:

```
print(numbers[17])
```

приводит к возникновению ошибки:

```
IndexError: index out of range
```

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]`.

Приведённый ниже 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.

Использование срезов для изменения элементов в заданном диапазоне

Для изменения целого диапазона элементов списка можно использовать срезы. Например, если мы хотим перевести на русский язык названия фруктов `'banana', 'cherry', 'kiwi'`, то это можно сделать с помощью среза.

Приведённый ниже код:

```
fruits = ['apple', 'apricot', 'banana', 'cherry', 'kiwi', 'lemon', 'mango']
fruits[2:5] = ['банан', 'вишня', 'киви']

print(fruits)
```

выводит:

```
['apple', 'apricot', 'банан', 'вишня', 'киви', 'lemon', 'mango']
```

Операция конкатенации + и умножения на число *

Мы можем применять операторы `+` и `*` для списков подобно тому, как мы это делали со строками.

Приведённый ниже 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]
```

Для генерации списков, состоящих строго из повторяющихся элементов, умножение на число — самый короткий и правильный метод.

Мы также можем использовать расширенные операторы `+=` и `*=` при работе со списками.

Приведённый ниже код:

```
a = [1, 2, 3, 4]
b = [7, 8]
a += b   # добавляем к списку a список b
b *= 5   # повторяем список b 5 раз

print(a)
print(b)
```

выводит:

```
[1, 2, 3, 4, 7, 8]
[7, 8, 7, 8, 7, 8, 7, 8, 7, 8]
```

Встроенные функции sum(), min(), max()

Встроенная функция `sum()` принимает в качестве параметра список чисел и вычисляет сумму его элементов.

Приведённый ниже код:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print('Сумма всех элементов списка =', sum(numbers))
```

выводит:

```
Сумма всех элементов списка = 55
```

Встроенные функции `min()` и `max()` принимают в качестве параметра список и находят минимальный и максимальный элементы соответственно.

Приведённый ниже код:

```
numbers = [3, 4, 10, 3333, 12, -7, -5, 4]
print('Минимальный элемент =', min(numbers))
print('Максимальный элемент =', max(numbers))
```

выводит:

```
Минимальный элемент = -7
Максимальный элемент = 3333
```

Отличие списков от строк

Несмотря на всю схожесть списков и строк, есть одно очень важное отличие: строки — **неизменяемые** объекты, а списки – **изменяемые**.

Приведённый ниже код:

```
s = 'abcdefg'
s[1] = 'x'    # пытаемся изменить 2 символ (по индексу 1) строки
```

приводит к возникновению ошибки:

```
object does not support item assignment
```

Приведённый ниже 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]
```

Запомни: изменять отдельные символы строк нельзя, однако можно изменять отдельные элементы списков. Для этого используем индексатор и оператор присваивания.

Happy Pythoning!

Шаг 2

Используя индексатор, дополните приведённый ниже код так, чтобы он вывел 17-ый (по порядку) элемент списка `primes`.

Чтобы решить это задание откройте https://stepik.org/lesson/296419/step/2

Шаг 3

Используя индексатор, дополните приведённый ниже код так, чтобы он вывел последний элемент списка `primes`.

Чтобы решить это задание откройте https://stepik.org/lesson/296419/step/3

Шаг 4

Используя срезы, дополните приведённый ниже код так, чтобы он вывел первые 6 элементов списка `primes`.

Примечание. Результатом вывода должна быть строка `[2, 3, 5, 7, 11, 13]`.

Чтобы решить это задание откройте https://stepik.org/lesson/296419/step/4

Шаг 5

Дополните приведённый ниже код так, чтобы он вывел **сумму** минимального и максимального элементов списка `numbers`.

Чтобы решить это задание откройте https://stepik.org/lesson/296419/step/5

Шаг 6

Дополните приведённый ниже код так, чтобы он вывел среднее арифметическое элементов списка `evens`.

Чтобы решить это задание откройте https://stepik.org/lesson/296419/step/6

Шаг 7

Дополните приведённый ниже код так, чтобы элемент списка имеющий значение `Green` заменился на значение `Зеленый`, а элемент `Violet` – на `Фиолетовый`. Далее необходимо вывести полученный список.

Чтобы решить это задание откройте https://stepik.org/lesson/296419/step/7

Шаг 8

Дополните приведённый ниже код так, чтобы он вывел "перевёрнутый" список `languages` (то есть элементы будут идти в обратном порядке).

Примечание. Для вывода списка воспользуйтесь функцией `print()`.

Чтобы решить это задание откройте https://stepik.org/lesson/296419/step/8

Шаг 9

Используя операторы конкатенации (`+`) и умножения списка на число (`*`), дополните приведённый ниже код так, чтобы он вывел следующий список:

```
[1, 2, 3, 1, 2, 3, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 8, 9, 10, 11, 12, 13]
```

Чтобы решить это задание откройте https://stepik.org/lesson/296419/step/9