Университет Сириус Колледж

Основы программирования. Лекция 2

Структуры данных Python. Списки.

Университет Сириус Колледж

Структуры данных

- Структуры данных описывают точку зрения пользователя на представление данных.
- Любая структура данных имеет ограниченный набор операций, которые на этой структуре можно выполнять.
- Структура данных имеет ряд условных правил (ограничений), определяющих соответствие данных этой структуре

Университет Сириус Колледж

Структуры данных в Python

В языке Python существует ряд основных структур данных:

- Списки (lists);
- Кортежи (tuples);
- Словари (dictionaries);
- Множества (sets).

Университет Сириус Колледж

Список (list)

- Список служит для того, чтобы хранить объекты (данные) в определенном порядке, особенно если порядок или содержимое могут изменяться.
- Списки можно изменять: можно добавить или удалить элементы, а также перезаписать существующие.
- Примеры списков:

```
>> empty_list  = []
>> numbers = [1, 2, 3, 4, 5]
```

Также список можно объявить с помощью `list()` :

```
>> new_empty_list = list()
```

Университет Сириус Колледж

Инициализация списка

Создать список определенного размера и сразу заполнить его значениями можно с помощью оператора умножения (`*`)

```
>> zeros = [0] * 5
>> zeros
[0, 0, 0, 0, 0]
>> ones = [1] * 5
>> ones
[1, 1, 1, 1, 1]
>> repetition = [1, 2, 3] * 3
>> repetition
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Университет Сириус Колледж

Создание списков

Список также можно получить из других итерируемых типов данных.

Например, из строки:

```
>> list('cat')
['c', 'a', 't']
```

Или из кортежа (tuple):

```
>> new_tuple = ('1', '2', '3')
>> list(new_tuple)
['1', '2', '3']
```

Создание списков

Также многие стандартные функции возвращают список.

Например, функция `split()` применительно к строке разбивает эту строку на сегменты, по указанному пользователем разделителю, или же по пробелу, если таковой не указан:

```
>> today = '25/07/2021'
>> today.split('/')
['25', '07', '2021']
```

Университет Сириус Колледж

Получение элементов списка

Получить элемент списка можно, указав его смещение:

```
>> numbers = ['0', '5', '10']
>> numbers[1]
'5'
```

Не забывайте, что индексация элементов начинается с нуля, а отрицательные индексы отсчитываются с конца строки:

```
>> numbers[-1]
'10'
```

Университет Сириус Колледж

Вложенные списки (nested lists)

В качестве значений списки могут содержат другие списки. Например, у нас есть список маленьких птиц, а есть список птиц побольше.

Объявим список `all_birds`, содержащий всех наших птиц:

```
>> small_birds = ['hummingbird', 'sparrow']
>> bigger_birds = ['pigeon', 'crow']

>> all_birds = [small_birds, bigger_birds]
>> all_birds
[['hummingbird', 'sparrow'], ['pigeon', 'crow']]
```

Университет Сириус Колледж

Инициализация вложенных списков

Вложенные списки можно инициализировать аналогично обычным спискам.

```
>> nested = [[0, 1], [2, 3]]
```

Университет Сириус Колледж

Изменение элемента списка

Элемент списка можно изменить, также обратившись к нему по его смещению:

```
>> words = ['hello', 'world']
>> words[1] = 'python'
>> words
['hello', 'python']
```

Университет Сириус Колледж

Извлечение элементов с помощью диапазона смещений

Из списка можно извлечь последовательность, использовав диапазон смещений:

```
>> numbers = [0, 1, 2, 3, 4, 5]
>> numbers[0:2]
[0, 1]
>> numbers[::-1]
[5, 4, 3, 2, 1, 0]
```

Университет Сириус Колледж

Добавление элементов в список

- С помощью метода `append()`:

```
>> numbers.append(6)
[0, 1, 2, 3, 4, 5, 6]
```

- С помощью оператора `+=`:

```
>> numbers += '7'
[0, 1, 2, 3, 4, 5, 6, 7]
```

- С помощью метода `extend()` (добавление списка к существующему):

```
>> numbers.extend([8,  9])
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- С помощью метода `insert()`:

```
>> numbers.insert(0,  -1)
[-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Университет Сириус Колледж

Удаление элементов из списка

- С помощью `del` (удаление по индексу):

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

- С помощью `remove()` (удаление по значению):

```
>> numbers.remove(8)
[0, 1, 2, 3, 4, 5, 6, 7, 9]
```

Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение `ValueError`

- С помощью `clear()`:

```
>> numbers.clear()
[]
```

Университет Сириус Колледж

Удаление элементов из списка

- С помощью метода `pop([index])`, возвращающей элемент по указанному индексу и удаляющей его из списка:

```
>> numbers.pop(0)
0
>> numbers
[1, 2, 3, 4, 5, 6, 7, 9]
```

Параметр `index` по умолчанию равен `-1`, поэтому функция `pop()` возвращает последний элемент списка.

```
>> numbers.pop()
9
>> numbers
[1, 2, 3, 4, 5, 6, 7]
```

Университет Сириус Колледж

Поиск элемента в списке

С помощью метода `index()`:

```
>> numbers.index(7)
6
```

> *Если элемент не найден, то будет выведено исключение `ValueError`.*

С помощью метода `count()`:

```
>> numbers.count(3)
1
```

Сортировка списка

С помощью метода `sort([key, reverse])` или функции `sorted(list, [key, reverse])`.

Если параметры `key` и `reverse` не указаны, то сортируются элементы списка по неубыванию.

Параметр `reverse` указывает на то, что список должен быть отсортирован в обратном порядке.

```
>> numbers = [3, 1, 2, 4, 5]
>> numbers.sort()
>> numbers
[1, 2, 3, 4, 5]
```

Университет Сириус Колледж

Сортировка списка

Параметр `key` указывает на то, что список должен быть отсортирован по функции `key`.

Функция `key` должна принимать один аргумент и возвращать число.

```
>> numbers = [3, 1, 2, 4, 5]
>> numbers.sort(key=lambda x: -x)
>> numbers
[5, 4, 3, 2, 1]
```

Университет Сириус Колледж

Функции списков

- `len()` - длина списка

```
>> numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>> len(numbers)
10
```

- `max()` - максимальное значение

```
>> numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>> max(numbers)
9
```

Функции списков

- `min()` - минимальное значение

```
>> numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>> min(numbers)
0
```

- `sum()` - сумма значений

```
>> numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>> sum(numbers)
45
```

Университет Сириус Колледж

Списковое включение (List comprehension)

Списковое включение — это некий синтаксический сахар, позволяющий упростить генерацию последовательностей (списков, множеств, словарей, генераторов).

```
новый_список = [«операция» for «элемент списка» in «список»]
```

- операция подразумевает некие действия, которые вы собираетесь применить к каждому элементу списка;
- элемент списка — каждый отдельный объект списка;
- список — последовательность, элементы которой вы планируете подвергнуть операции (это не обязательно должен быть `list`, подойдет любой итерируемый объект).

```
>> old_prices = [120, 550, 410, 990]
>> discount = 0.15
>> new_prices = [int(product * (1 - discount)) for product in old_prices]
```

Университет Сириус Колледж

Списковое включение
Условие в конце включения

```
новый_список = [«операция» for «элемент списка» in «список» if «условие»]
```

Такой вариант использования условий позволяет отсечь часть элементов итератора. Новый список будет короче первоначального. По сути, к той же конструкции, которая приведена выше, добавляется условие if.

```
>> numbers = [121, 544, 111, 99, 77]
>> number11 = [num for num in numbers if num % 11 == 0]
>> number11
[121, 99, 77]
```

Следует обратить внимание, что условие может быть только одно (т. е. здесь невозможно использовать `elif`, `else` или другие `if`, как мы могли бы сделать в циклах).

Университет Сириус Колледж

Списковое включение
Условие в начале включения

Если требуется не фильтрация данных по какому-то критерию, а изменение типа операции над элементами последовательности, условия могут использоваться в начале генератора списков.

```
новый_список = [«операция» if «условие» for «элемент списка» in «список»]
```

> *В отличие от предыдущего типа условий, здесь оно может дополняться вариантом `else` (но `elif` и тут невозможен).*

```
>> from string import ascii_letters
>> letters = 'hытφтгцзqπ'
>> is_eng = [
...     f'{letter}-ДА' if letter in ascii_letters else f'{letter}-НЕТ'
...     for letter in letters
... ]
>> print(is_eng)
['h-ДА', 'ы-НЕТ', 't-ДА', 'φ-НЕТ', 'т-НЕТ', 'r-ДА', 'ц-НЕТ', 'з-НЕТ', 'q-ДА',
```

Университет Сириус Колледж

Сложные списковые включения

```
>> words = ['Я', 'изучаю', 'Python']
>> letters = [letter for word in words for letter in word]
>> letters
['Я', 'и', 'з', 'у', 'ч', 'а', 'ю', 'P', 'y', 't', 'h', 'o', 'n']
```

```
>> table = [[x * y for x in range(1, 6)] for y in range(1, 6)]
>> table
[[1, 2, 3, 4, 5],
 [2, 4, 6, 8, 10],
 [3, 6, 9, 12, 15],
 [4, 8, 12, 16, 20],
 [5, 10, 15, 20, 25]]
```