TOMSK POLYTECHNIC UNIVERSITY
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Системный анализ процессов химической технологии

Лабораторная работа №2
Структуры данных: списки

Вячеслав Алексеевич Чузлов,
к.т.н., доцент ОХИ ИШПР

6 февраля 2023 г.

TOMSK POLYTECHNIC UNIVERSITY

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Списки

Списки

- Списки в Python – наиболее гибкая разновидность объектов упорядоченных коллекций.
- Списки могут содержать объекты любого типа: строки, числа или другие списки.
- Списки **можно изменять** на месте присваиванием по индексам, с использованием срезов, вызвав специальные методы или выполнив оператор удаления.

Операция	Описание
`a = []`	Пустой список
`a = [123, 'abc', 1.354, []]`	Четыре элемента: индексы 0…3
`a = ['Joe', 30.0, ['dev', 'prof']]`	Вложенные списки
`a = list('hello')`	Список элементов итерируемого объекта
`a = list(range(-5, 6))`	Список последовательных целых чисел
`a[i]`	Индекс
`a[i][j]`	Индекс индекса
`a[i:j]`	Срез
`len(a)`	Длина
`a1 + a2`	Конкатенация
`a * 3`	Повторение
`x in a`	Вхождение
`a.append(5)`	Добавление элемента в конец списка
`a.extend([10, 20, 30])`	Добавление списка в конец исходного списка

Базовые операции со списками

- Списки являются **последовательностями**, поэтому поддерживают многие операции, характерные для строк.
- Например, для списков определены операторы `+` и `*`. Данные операторы, также как и в случае со строками, означают конкатенацию и повторение, только возвращают в качестве результата новый список, а не строку.

```
>>> len([1, 2, 3, 4, 5])                # Длина
5

>>> [1, 2, 3, 4, 5] + [6, 7, 8, 9, 10]  # Конкатенация
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> ['Hi!'] * 5                         # Повторение
['Hi!', 'Hi!', 'Hi!', 'Hi!', 'Hi!']
```

Итерация по спискам

В общем смысле для списков определены все операции над последовательностями, в том числе и инструменты итерации:

```
>>> 'banana' in ['banana', 'orange', 'apple']   # Проверка вхождения
True

>>> for fruit in ['banana', 'orange', 'apple']: # Итерация
...     print(fruit, end=' ')
...
banana orange apple
```

Оператор цикла `for` проходит (итерируется) по всем элементам в любой последовательности (итерируемом объекте) слева направо, выполняя операторы для каждого из них.

Генераторы списков (list comprehension)

Генераторы списков – это способ создания нового списка с применением выражения к каждому элементу последовательности (по факту в любом итерируемом объекте).

```
>>> res = [c * 4 for c in 'HELLO']

>>> res
['HHHH', 'EEEE', 'LLLL', 'LLLL', 'OOOO']
```

- Генераторы списков записываются более кратко и выполняются чуть быстрее.
- В сложных случаях лучше использовать цикл `for` из-за его более высокой читаемости.

```
>>> res = []

>>> for c in 'HELLO':
...     res.append(c * 4)
...

>>> res
['HHHH', 'EEEE', 'LLLL', 'LLLL', 'OOOO']
```

Индексация и срезы

- Индексация и срезы для списков работают аналогично тому, как это было описано для объектов строк.
- Результатом индексации списка может быть объект любого типа, находящийся по указанному индексу, тогда как срезы всегда возвращают новый объект списка.

```
>>> fruits = ['banana', 'orange', 'apple']

>>> fruits[2]            # Индексы начинаются с нуля
'apple'

>>> fruits[-2]           # Отрицательные индексы отсчитываются справа
'orange'

>>> fruits[1:3]          # Срезы получают сегменты
['orange', 'apple']

>>> fruits[-1]           # Результат среза всегда новый список
['apple']
```

Вложенность списков

- Внутри списков могут содержаться вложенные списки или объекты других типов.
- Матрицы в Python можно представить в виде вложенных списков. Пример матрицы 3×3:

```
>>> matrix = [[10, 20, 30], [40, 50, 60], [70, 80, 90]]
```

- Если указать один индекс, то будет получена целая строка, а при указании двух индексов будет возвращен элемент строки:

```
>>> matrix[1]
[40, 50, 60]

>>> matrix[2][0]
70

>>> matrix = [[10, 20, 30],
...           [40, 50, 60],
...           [70, 80, 90]]

>>> matrix[1][1]
50
```

Изменение списков

- Так как списки – **изменяемый** тип объектов, для них определены операции, которые могут модифицировать объект списка **на месте**.
- Операции модифицирования списков изменяют объект списка напрямую, перезаписывая его старое значение, без необходимости создания новой копии, как в случае работы со строками.

Присваивание по индексам и срезам

Содержимое списка может быть изменено присваиванием значения либо отдельному элементу (по его индексу), либо целому сегменту (по срезу):

```
>>> food = ['burger', 'pizza', 'buritto']

>>> food[1] = 'toast'  # Присваивание по индексу

>>> food
['burger', 'toast', 'buritto']

>>> food[:2] = ['need', 'more']  # Присваивание срезу

>>> food
['need', 'more', 'buritto']
```

Вызовы методов списков

- Подобно строкам, списки имеют набор специфичных методов, многие из которых ведут к изменению исходного списка на месте:

```
>>> a = ['eat', 'more', 'SPAM']

>>> a.append('please')

>>> a
['eat', 'more', 'SPAM', 'please']

>>> a.sort()  # Сортировка элементов списка ('S' < 'e')

>>> a
['SPAM', 'eat', 'more', 'please']
```

- Наиболее распространенный метод – `append()` добавляет объект в конец списка.
- Эффект выполнения выражения `a.append(x)` аналогичен `a + [x]` с одним принципиальным отличием: первый вариант изменяет a на месте, а второй вариант создает новый объект списка.
- Метод `sort()` упорядочивает элементы в списке.

Дополнительные сведения о методе `sort()`

- В методе `sort()` аргумент `reverse` позволяет производить сортировку в порядке убывания вместо возрастания, а параметр `key` задает функцию с одним аргументом, которая возвращает значение для использования при сортировке.

```
>>> a = ["abc", "ABD", "aBe"]

>>> a.sort()                                # Сортировка со смешанным регистром

>>> a
['ABD', 'aBe', 'abc']

>>> a = ["abc", "ABD", "aBe"]

>>> a.sort(key=str.lower)                   # Приведение к нижнему регистру

>>> a
['abc', 'ABD', 'aBe']

>>> a.sort(key=str.lower, reverse=True)  # Изменение порядка сортировки

>>> a
['aBe', 'ABD', 'abc']
```

TOMSK POLYTECHNIC UNIVERSITY

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Пример

Пример

По имеющимся исходным данным определите состав потока в объемных и мольных долях, используя следующие формулы:

$$\varphi_i = \frac{\frac{\omega_i}{\rho_i}}{\sum_{i=1}^{n} \frac{\omega_i}{\rho_i}} \qquad \chi_i = \frac{\frac{\omega_i}{M_i}}{\sum_{i=1}^{n} \frac{\omega_i}{M_i}}$$

где φ_i – объемная доля i-го компонента; χ_i – мольная доля i-го компонента; ω_i – массовая доля i-го компонента; ρ_i – плотность i-го компонента; M_i – молярная масса i-го компонента; n – число компонентов в системе; i – индекс компонента в системе.

Пример

Исходные данные:

Параметр	C_1	C_2	C_3	iC_4	nC_4	iC_5	nC_5	C_6
ω_i	0.1	0.1	0.1	0.4	0.2	0.05	0.03	0.02
ρ_i, г/см^3	0.416	0.546	0.585	0.5510	0.6	0.616	0.6262	0.6594
M_i, г/моль	16	30	44	58	58	72	72	86

Пример

```python
mass_fractions = [.1, .1, .1, .4, .2, .05, .03, .02]
densities = [0.416, 0.546, 0.585, 0.5510, 0.6, 0.616, 0.6262, 0.6594]
mol_mass_list = [16, 30, 44, 58, 58, 72, 72, 86]

volume_fractions = [mf / d for mf, d in zip(mass_fractions, densities)]
s = sum(volume_fractions)
volume_fractions = [vof / s for vof in volume_fractions]

volume_fractions = []  # Alternative
for mf, d in zip(mass_fractions, densities):
    volume_fractions.append(mf / d)

s = sum(volume_fractions)
for i in range(len(volume_fractions)):
    volume_fractions[i] /= s

for vof in volume_fractions:  # вывод результата
    print(f'{vof:.4f}', end=' ')
```

```
0.1326 0.1010 0.0943 0.4004 0.1838 0.0448 0.0264 0.0167
```

TOMSK POLYTECHNIC UNIVERSITY
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Контакты

Вячеслав Алексеевич Чузлов,
к.т.н., доцент ОХИ ИШПР

Учебный корпус №2, ауд. 136
chuva@tpu.ru
+7-962-782-66-15

Благодарю за внимание!