Программирование, лекция 5. Списки

Кафедра ИУ7 МГТУ им. Н. Э. Баумана, 2022 год

Другие стандарты схем

- IDEF (I-CAM DEFinition или Integrated DEFinition) - для моделирования широкого спектра сложных систем в различных разрезах
- UML (Unified Modeling Language) - открытый стандарт, использующий графические обозначения для создания абстрактной модели системы
- ARIS (Architecture of Integrated Information Systems) - методология для моделирования бизнес-процессов организаций

Способы классификации типов данных

- простые и сложные (составные) (массивы, записи, файлы)
- скалярные и нескалярные (агрегатные типы данных)
- самостоятельные и зависимые (например, ссылки)

Коллекции данных

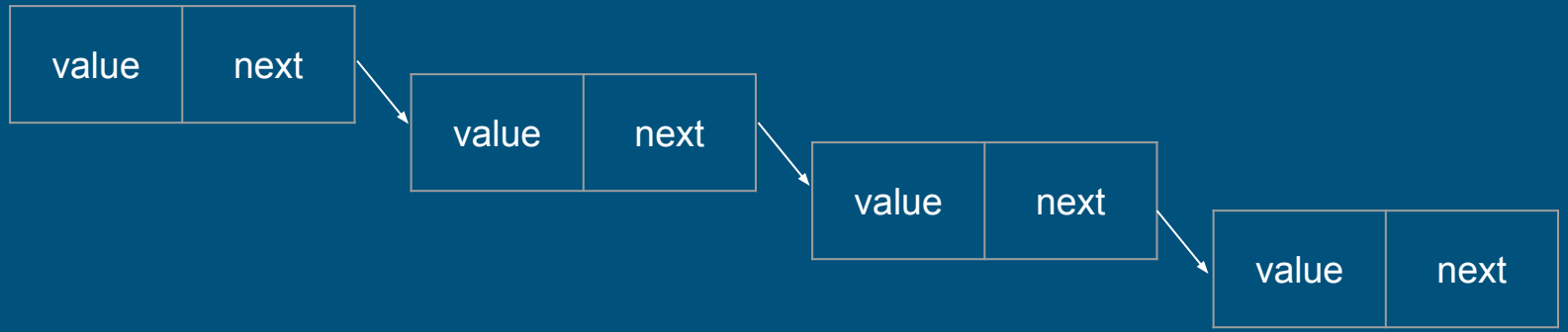
Коллекция - объект, содержащий в себе набор значений одного или различных типов и позволяющий обращаться к этим значениям.

Виды:

- массив
  - одномерный - вектора
  - двумерный - матрица
  - многомерный
- список
- ассоциативный массив, очередь, стек, множество...

Массивы и списки

0	1	2	3	4	5	...
value0	value1	value2	value3	value4	value5	...



Вычислительная сложность

Понятие в теории алгоритмов, обозначающее функцию зависимости объёма работы (по времени или памяти), которая выполняется некоторым алгоритмом, от размера входных данных.

O(n) - линейная

$O(n^2)$ - квадратичная

O(log n) - логарифмическая

O(n * log n)

Сравнение вычислительной сложности работы с массивами и списками

Просмотр всех элементов (поиск) - O(n)

Доступ к элементу по индексу - O(1) vs O(n)

Вставка O(n) vs O(1)

Удаление O(n) vs O(1)

Встроенные структуры данных Python

Неизменяемые (immutable):

- range
- tuple (кортеж) - (...)

Изменяемые (mutable):

- list (список) - [...]
- set (множество)
- dict (словарь) - {...}

Списки в Python

Создание списка:

- numbers = [1, 1, 2, 3, 5, 8, 13, 21]
- a = list()

Индексация элементов - с 0 по порядку.

Обращение к элементам:

- numbers[4] # 5
- numbers[-1] # 21

Функции для работы со списками

- all() - возвращает True, если все элементы истинны или список пуст
- enumerate(iterable, start=0) - возвращает ***перечисляемый*** объект - кортежи (индекс, значение)
- len(s) - количество элементов списка
- list([iterable]) - создание списка на основе итерируемого объекта, например, a = list(range(10))
- max(iterable)
- min(iterable)
- print()
- reversed(seq) - возвращает итератор. Не создаёт копию последовательности. b = list(reversed(a)). Подходит для range() и других объектов, удовлетворяющих требованиям.
- sorted(iterable, key = None, reverse = False)
- sum(iterable)

Некоторые понятия ООП

Класс — некоторый шаблон для создания объектов, обеспечивающий начальные значения состояния: инициализация полей-переменных и реализация поведения методов.

Объект — это экземпляр с собственным состоянием этих свойств.

Поле - некоторое «свойство», или атрибут, какого-то объекта (переменная, являющаяся его частью). Объявляется в классе.

Метод - функция объекта, которая имеет доступ к его состоянию (полям). Реализуется в классе.

Методы списков

- append(x) - добавление элемента x в конец списка
- extend(iterable) - расширение списка с помощью итерируемого объекта
- insert(i, x) - вставка x в i-ю позицию. Если i за границами списка, то вставка происходит в конец/начало списка
- remove(x) - удаляет первый элемент со значением x
- pop([i]) - удаляет элемент в позиции i. Если аргумент не указан, удаляется последний элемент списка
- clear() - удаляет все элементы из списка
- index(x[, start[, end]]) - возвращает индекс (с 0) первого элемента, равного x
- count(x) - возвращает количество вхождений x в список
- sort(key=None, reverse=False) - сортировка списка
- reverse() - разворачивает список (переставляет элементы в обратном порядке)
- copy() - создание “мелкой” копии

Операторы для работы со списками

- “+” - конкатенация списков. Аналогично extend, но только для списков
- “*” - “умножение” списка. list * число - увеличить длину списка в n раз, скопировав элементы
- “in” - принадлежность значения списку: 5 in [3,5,7]
- del - удаление переменной или элемента
- “==” - сравнение списков на совпадение элементов с учётом порядка
- “>” “>=” “<” “<=” - сравнение списков с учётом лексикографического порядка элементов
- for i in list: ...

Срезы

[start:stop:step] - возвращает элементы списка, начиная с индекса start до stop с шагом step.

a[:] - все элементы списка

a[5:] - с элемента с индексом 5 до конца

a[:2] - элементы 0 и 1

a[::-1] - разворачивание списка

Создание списков

```
a = [] # пустой список
a = [0] * 10     # список фиксированного размера, инициализированный
                 #начальными значениями
# списковые включения (list comprehension)
a = [i*i for i in range(10)]
a = [i for i in range(10) if i % 2 == 0]
```

Способы ввода списков

```
# 1. быстро, плохо:

a = list(map(int, input('Введите массив (в одну строку через пробел): ').split()))


# 2. по элементам с указанием размера:

n = int(input('Введите размер массива:'))

a = [0] * n

for i in range(n):

    a[i] = int(input('Введите {}-й элемент: ').format(i+1))
```

Способы ввода списков (продолжение)

```
# по элементам без ввода размера:
a = []
i = 0
while True:
    i += 1
    el = input('Введите {}-й элемент: ').format(i)
    if el:
        a.append(int(el))
    else:
        break
```

Вывод списка

```
print(a) # плохо. подходит только для отладки


# лучше - с форматированием и указанием номеров
print('\nВведённый массив:')
for (i, el) in enumerate(a):
    print('{:2}-й элемент: {}'.format(i + 1, el))
```