Методичка для студентов

Python: словари (dict). Подсчет частот и поиск лидера

Уровень: базовый (после списков и циклов). Формат: 1 занятие + практика.

Цель урока: научиться хранить и быстро находить данные по ключу, работать с парами ключ-значение, перебирать словарь и решать типовые задачи: поиск, группировка, подсчет частоты.

Что вы должны уметь после урока:

- создавать словарь и добавлять/обновлять элементы;
- безопасно получать значение (get) и проверять наличие ключа (in);
- перебирать словарь через keys(), values(), items();
- решать задачу 'частоты' (частота элементов в списке / строке);
- находить 'лучший' элемент по нескольким критериям (max + tie-break).

Предпосылки: вы уже знаете list, циклы for/while, условия if/elif/else, функции print/input.

1. Что такое словарь (dict)

Словарь - это структура данных, которая хранит пары ключ -> значение.

Ключ отвечает на вопрос: по чему искать? Значение: что хранить?

Примеры:

- name -> age ("Ali" -> 18)
- login -> password_hash ("icdfh" -> "...hash...")
- product_id -> product_object (42 -> {title, price, category})
- country -> capital ("KZ" -> "Astana")

Главная идея: доступ к данным идет не по индексу, как в списке, а по ключу.

Сравнение

Структура	Как хранит	Как получить элемент	Когда удобно
list	упорядоченные элементы	arr[i] (индекс)	когда важен порядок, просто список значений
dict	пары ключ-значение	d[key] (ключ)	когда нужен быстрый поиск по имени/id/коду

2. Создание словаря

Пустой словарь:

```
d = {}
# или
d = dict()
```

Словарь с данными:

```
ages = {
    "Ali": 18,
    "Dana": 21,
    "Mansur": 17
}
```

3. Ключи: что можно использовать

Ключ должен быть неизменяемым (то есть таким, который нельзя поменять 'на месте').

Обычно ключи делают строками или числами.

Можно: str, int, float, bool, tuple. Нельзя: list, dict, set.

```
d = {
    1: "one",
    "2": "two",
    (1, 2): "tuple key"
}
```

4. Доступ к значениям

4.1 Через квадратные скобки

Так можно, когда вы уверены, что ключ существует:

```
print(ages["Ali"])  # 18
```

Если ключа нет, Python выдаст ошибку KeyError.

4.2 Безопасно через get

Метод get возвращает значение, если ключ есть, и default (по умолчанию None), если ключа нет.

```
print(ages.get("Sanya"))        # None
print(ages.get("Sanya", 0))     # 0 - значение по умолчанию
```

> Правило для практики:

- d[key] - используем, когда ключ точно есть;
- d.get(key) - используем, когда ключ может отсутствовать.

5. Добавление и обновление

Добавление нового элемента:

```
ages["Sanya"] = 20
```

Обновление существующего:

```
ages["Ali"] = 19
```

6. Удаление

Удалить по ключу:

```
del ages["Dana"]
```

Удалить и получить значение:

```
x = ages.pop("Mansur")
print(x)
```

7. Проверка наличия ключа

```
if "Ali" in ages:
    print("Есть")
else:
    print("Нет")
```

8. Перебор словаря

Словарь можно перебирать тремя основными способами.

8.1 По ключам (по умолчанию)

```
for name in ages:
    print(name)
```

8.2 По значениям

```
for age in ages.values():
    print(age)
```

8.3 По парам ключ-значение (самое полезное)

```
for name, age in ages.items():
    print(name, "=>", age)
```

9. Частая ошибка: дубли ключей (почему остается только последний)

В словаре ключи уникальны. Если вы повторяете ключ, вы перезаписываете прошлое значение.

Плохой пример:

```
a = {
    "id": 1, "name": "Ali",
    "id": 2, "name": "Qwerty",
    "id": 3, "name": "Asd"
}
print(a)
```

Почему так: ключи 'id' и 'name' повторяются. Остаются последние значения: id=3, name='Asd'.

Как правильно хранить несколько людей

Вариант A: список словарей

```
users = [
    {"id": 1, "name": "Ali"},
    {"id": 2, "name": "Qwerty"},
    {"id": 3, "name": "Asd"},
]
```

Вариант B: словарь по id

```
users = {
    1: {"name": "Ali"},
    2: {"name": "Qwerty"},
```

```
    3: {"name": "Asd"},
}
print(users[2]["name"])  # Qwerty
```

10. Практика (с подробным разбором решений)

Правило: сначала читаем условие, затем делаем 'скелет' (переменные и цикл), потом дописываем детали.

Практика 1. Телефонная книга

Задача: создать словарь phonebook, добавить минимум 3 контакта (имя -> телефон) и вывести все пары 'Имя => телефон'.

Решение:

```
phonebook = {}
phonebook["Ali"] = "+7701 000 00 01"
phonebook["Dana"] = "+7702 000 00 02"
phonebook["Mansur"] = "+7703 000 00 03"

for name, phone in phonebook.items():
    print(name, "=>", phone)
```

Пояснение:

- phonebook.items() возвращает пары (ключ, значение).
- Мы распаковываем их в name и phone.

Практика 2. Безопасный поиск через get

Задача: пользователь вводит имя. Если контакт есть - вывести телефон. Если нет - вывести 'Нет контакта'.

Решение:

```
phonebook = {"Ali": "111", "Dana": "222"}
name = input("Имя: ").strip()

phone = phonebook.get(name)  # None, если ключа нет
if phone is None:
    print("Нет контакта")
else:
    print("Телефон:", phone)
```

Пояснение:

- get возвращает None, если ключ отсутствует.
- strip() убирает лишние пробелы в начале/конце ввода.

Практика 3. Частота чисел (самый важный шаблон dict)

Задача: дан список nums. Посчитать, сколько раз встречается каждое число.

```
nums = [1, 2, 2, 3, 3, 3, 5, 5]
```

Идея: делаем словарь freq, где ключ это число, а значение это счетчик.

Решение (через get):

```
nums = [1, 2, 2, 3, 3, 3, 5, 5]
freq = {}

for x in nums:
    freq[x] = freq.get(x, 0) + 1

print(freq)  # {1: 1, 2: 2, 3: 3, 5: 2}
```

Разбор строки freq[x] = freq.get(x, 0) + 1:

- freq.get(x, 0) - если x уже есть, берем счетчик; если нет, считаем 0.
- прибавляем 1 - потому что встретили x еще один раз.
- записываем обратно в freq[x].

Аналогичное решение (без get, через if):

```
freq = {}
for x in nums:
    if x in freq:
        freq[x] += 1
    else:
        freq[x] = 1
```

Практика 4. Лидер частоты (max + tie-break)

Задача: найти число, которое встречается чаще всего. Если частота одинаковая - выбрать самое маленькое число.

```
nums = [5,5,2,2,2,3,3,5,1,1,1]
```

Шаги решения:

- Сначала считаем частоты в freq.
- Потом проходим по freq.items() и выбираем лучшую пару.
- Храним best_num и best_count.

Решение:

```
nums = [5,5,2,2,2,3,3,5,1,1,1]
```

```
freq = {}
for x in nums:
    freq[x] = freq.get(x, 0) + 1

best_num = None
best_count = -1

for num, count in freq.items():
    if count > best_count or (count == best_count and (best_num is None or num < best_num)):
        best_num = num
        best_count = count

print("freq:", freq)
print("most frequent:", best_num, "(count=", best_count, ")", sep="")
```

Разбор практики 4 по шагам

После подсчета частот получаем, например:

```
freq = {5: 3, 2: 3, 3: 2, 1: 3}
```

Критерии выбора:

- больше count (частота) - лучше;
- если count одинаковый - меньше num - лучше.

Зачем best_num = None и best_count = -1?

- best_count = -1: первая реальная частота (>=1) точно станет лучше.
- best_num = None: защита от сравнения num < None на первом шаге.

Условие выбора:

```
if count > best_count or (count == best_count and (best_num is None or num < best_num)):
    ...
```

Мини-трассировка (пример)

Пары (5,3) -> (2,3) -> (3,2) -> (1,3):

Шаг	(num,count)	best_num	best_count	Почему
1	(5,3)	5	3	3 > -1
2	(2,3)	2	3	частоты равны, 2 < 5
3	(3,2)	2	3	частота меньше
4	(1,3)	1	3	частоты равны, 1 < 2

Итог: best_num = 1, best_count = 3.

Практика 5. Группировка по категориям (dict + list)

Задача: собрать grouped, где ключ - категория, значение - список названий товаров.

```
products = [
    {"title": "Iphone", "category": "mobile"},
    {"title": "Macbook", "category": "laptop"},
    {"title": "Pixel", "category": "mobile"},
]
```

Решение:

```
grouped = {}

for p in products:
    cat = p["category"]
    title = p["title"]

    if cat not in grouped:
        grouped[cat] = []

    grouped[cat].append(title)

print(grouped)
```

Почему значение - список? Потому что в одной категории много товаров.

11. Мини-проект на закрепление (15-25 минут)

Нужно: category -> count, лидер категории, и вывод отчета.

Пример данных:

```
products = [
    {"title": "Iphone", "category": "mobile", "price": 500000},
    {"title": "Pixel", "category": "mobile", "price": 750000},
    {"title": "Macbook Air", "category": "laptop", "price": 800000},
    {"title": "Lenovo Legion", "category": "laptop", "price": 700000},
    {"title": "Ipad", "category": "tablet", "price": 400000},
]
```

Решение (пример):

```
cat_count = {}
for p in products:
    cat = p["category"]
    cat_count[cat] = cat_count.get(cat, 0) + 1
```

```
best_cat = None
best_count = -1
for cat, count in cat_count.items():
    if count > best_count or (count == best_count and (best_cat is None or cat < best_cat)):
        best_cat = cat
        best_count = count

print("Category counts:", cat_count)
print("Leader:", best_cat, "(count=", best_count, ")", sep="")
```

12. Частые ошибки и как их исправлять

KeyError при d[key]

- Используйте get или проверяйте key in d.

Дубли ключей

- Ключи уникальны. Если нужно несколько значений - храните список в качестве значения.

Тип ключа

- Ключ должен быть неизменяемым (str/int/tuple).

13. Домашнее задание (без решений)

Задача 1 - name -> age

Словарь из 6 людей. Вывести всех. По вводу имени - вывести возраст или 'не найдено'.

Задача 2 - частота символов

Частота символов строки (без пробелов). Топ-1: max count, tie-break по алфавиту.

Задача 3 - grouped

Собрать category -> [titles] и красиво вывести.

Задача 4 - grades

Словарь login -> score. По вводу логина: вывести score или предложить добавить.

Задача 5 - аналитика

Частоты nums, топ-1, и сортировка пар (num,count) по count убыванию.

Конец методички