

Словари и Множества в Python

Педантичное погружение в структуры данных `dict` и `set`.

Made with GAMMA

I. Введение и Мотивация: Почему Нам Нужны dict и set?

Проблема со списками (List)

Представьте, что вам нужно хранить информацию о студентах и их оценках. Если использовать только списки, вам потребуется несколько параллельных структур:

- Один список с именами (например, `['Alice', 'Bob']`).
- Другой список с оценками (например, `[90, 85]`).
- Необходимо постоянно следить, чтобы индексы соответствовали друг другу. Это **неудобно**, **нечитаемо** и легко приводит к ошибкам (Index Error или логическим ошибкам при рассинхронизации).

> "Словарь позволяет нам связывать данные напрямую, используя осмысленные метки, а не ненадежные числовые индексы."



Ключевая Мотивация

Словарь (**dict**) решает проблему ассоциативного хранения данных. Множество (**set**) решает проблему уникальности и быстрой проверки принадлежности.

Цели Лекции

Понять Концепции

Разобраться в фундаментальной структуре словаря (**ключ-значение**) и множества (**уникальные элементы**).

Изучить Инструменты

Научиться создавать, изменять, итеративно работать и применять ключевые методы `dict` и `set`.

Увидеть Практику

Ознакомиться с примерами, где эти структуры данных обеспечивают **эффективность** и **лаконичность** кода.

Made with GAMMA

II. Словари (dict) — Фундамент Ассоциативного Хранения

Словари (Dictionaries) - это неупорядоченная (в логическом смысле) коллекция произвольных объектов в формате **ключ: значение** с доступом по ключу.

Концепция и Аналогия

Словарь похож на **телефонную книгу**: имя (ключ) однозначно связано с номером (значением). Ключи должны быть **неизменяемыми** (строки, числа, кортежи). Значения могут быть любого типа.

Создание Словаря

- Литерал: `student = {'name': 'Alice', 'age': 20}`
- Через dict(): `student = dict(name='Alice', age=20)`
- Из списка кортежей: `dict([('name', 'Alice'), ('age', 20)])`

Важное Замечание об Упорядоченности

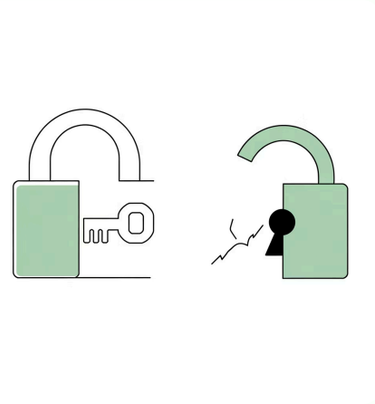
В Python 3.7+ (и CPython 3.6+) словари **сохраняют порядок** добавления ключей. Это очень удобно, но не следует полагаться на порядок, как на главный атрибут, как в списках.

Made with GAMMA

Ключевые Операции Со Словарями

Доступ и Изменение

- **Получение значения:** `dict[key]`.
- Внимание: Если ключа нет, вы получите ошибку `KeyError`.
- **Безопасное получение:** `.get(key, default)`.
- Если ключа нет, возвращается `default` (по умолчанию `None`). `print(student.get('grade', 'No grade'))`
- **Добавление/Изменение:** `dict[key] = value`.
- Если ключ существует, значение обновляется; если нет – создается новая пара.



Удаление Элементов

- `del dict[key]`**:** Удаляет пару по ключу. Вызывает `KeyError`, если ключа нет.
- `.pop(key, default)`**:** Удаляет ключ и **возвращает его значение**. Если ключ отсутствует, можно вернуть `default` вместо ошибки.
- `.clear()`**:** Полностью очищает словарь.

Представления Словаря (Views)

Методы `keys()`, `values()` и `items()` возвращают динамические представления, которые отражают текущее состояние словаря. Они очень эффективны для итерации.

.keys()

Представление всех ключей в словаре.

.values()

Представление всех значений.

.items()

Представление пар **(ключ, значение)**. Идеально для циклов: for k, v in dict.items():

Made with GAMMA

Практический Пример: Подсчет Частоты Слов (Word Count)

Одна из классических задач, где словарь незаменим: подсчитать, сколько раз каждое слово встречается в тексте. Здесь мы используем связку `dict[key] = dict.get(key, 0) + 1`.

Исходный Текст

```
text = "apple banana orange apple apple orange"
```

Логика Итерации

Мы проходим по каждому слову. Если слово уже есть в словаре, увеличиваем его значение на 1. Если нет, добавляем его со значением 1 (благодаря `.get(word, 0)`).

```
word_count = {}
for word in text.split():
    # Если слово уже есть, .get вернет его счетчик; иначе вернет 0.
    # Мы увеличиваем это значение на 1 и записываем обратно.
    word_count[word] = word_count.get(word, 0) + 1

print(word_count)
```

> Результат: `{'apple': 3, 'banana': 1, 'orange': 2}`

Метод `.update(other_dict)` позволяет элегантно объединять словари, перезаписывая совпадающие ключи значениями из второго словаря.

Made with GAMMA

III. Множества (set) — Коллекции Уникальных Элементов

Множества – это неупорядоченная коллекция, которая обеспечивает **уникальность** своих элементов и высокую скорость выполнения операции теории множеств.



Ключевая Концепция

- Множество не может содержать повторяющихся элементов. При попытке добавить дубликат, он будет проигнорирован.
- Как и ключи словаря, элементы множества должны быть **неизменяемыми** (например, нельзя добавить в множество список или другой словарь).
- **Аналогия:** Математическое множество или мешок с уникальными предметами. Порядок элементов не гарантирован.

Создание Множества

- **Литерал:** `fruits = {'apple', 'banana', 'orange'}`
- **Из итерируемого объекта:** `numbers = set([1, 2, 2, 3, 4])`. Дубликаты автоматически удаляются.
- **Пустое множество:**

Используйте только `empty_set = set()`.
(`{}` создает **пустой словарь, не множество**!)

Основные Операции Множества

→ **Добавление:** `.add(elem)`

→ **Безопасное Удаление:** `.discard(elem)` (Не вызывает ошибку, если элемента нет).

→ **Небезопасное Удаление:** `.remove(elem)` (Вызывает ошибку, если элемента нет).

→ **Проверка:** `if 'apple' in fruits:` – Это ОЧЕНЬ быстрая операция, что делает множества идеальными для поиска.

Made with GAMMA

IV. Теория Множеств: Мощные Инструменты

Главное преимущество `set` – возможность выполнять стандартные математические операции над множествами с помощью удобных операторов. Это незаменимо для сравнения коллекции данных.

Объединение (Union)

Все элементы из обоих множеств, без дубликатов. `a | b` или `a.union(b)`

Пересечение (Intersection)

Только общие элементы, присутствующие в обоих множествах. `a & b` или `a.intersection(b)`

Разность (Difference)

Элементы первого множества, которых нет во втором. `a - b` или `a.difference(b)`

Симметрическая Разность

Элементы, которые есть строго в одном из множеств (исключая общие). `a ^ b`

Практика: Нахождение Общих Элементов

Задача: найти общие элементы в двух списках. Если делать это циклами, это будет неэффективно. Множества решают это мгновенно.

```
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]

set1 = set(list1)
set2 = set(list2)

common_elements = set1 & set2

print(common_elements) # {4, 5}
```

Made with GAMMA

V. Сравнение Структур: dict vs set vs list

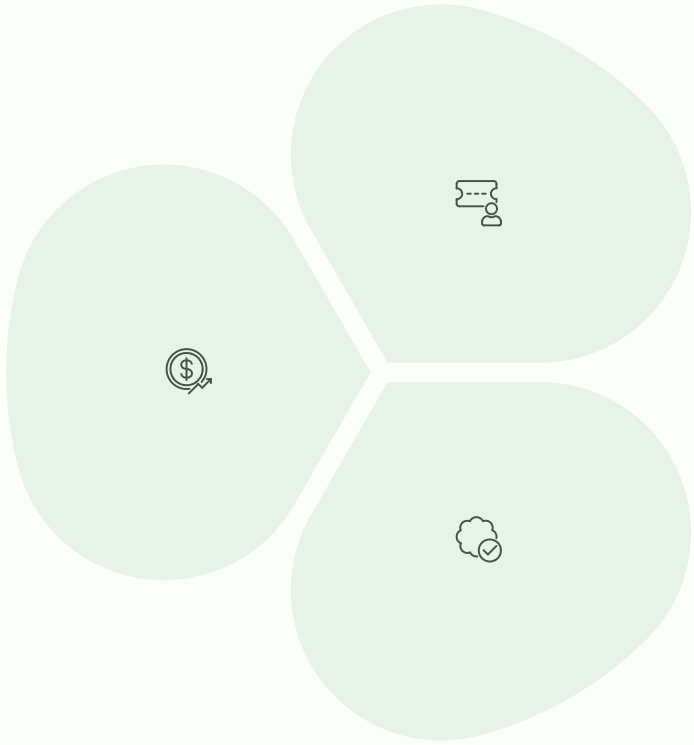
Для выбора оптимального инструмента важно понимать ключевые различия между основными коллекциями Python.

	list	set	dict
Упорядоченность	Да (по индексу)	Нет	Да (по порядку добавления, с Python 3.7+)
Доступ	По индексу (0, 1, 2...)	Быстрая проверка `in`	По неизменяемому ключу
Уникальность	Элементы могут повторяться	Все элементы **уникальны**	**Ключи** уникальны
Изменяемость	Изменяем (mutable)	Изменяем (mutable, но элементы - нет)	Изменяем (mutable)

Когда что использовать?

Список (list)

Нужен порядок элементов. Элементы могут повторяться. Доступ по индексу.



Словарь (dict)

Нужно хранить связанные данные (**ключ: значение**). Требуется быстрый поиск и доступ по имени (ключу).

Множество (set)

Нужна коллекция уникальных элементов. Требуется быстрое исключение дубликатов или выполнение операций теории множеств.

Made with GAMMA

VI. Закрепление: Практическая Задача

Задача: Учет Оценок Студентов

Создайте словарь `grades`, где ключами являются имена студентов (строка), а значениями – **списки** их оценок (числа).

- Напишите код, который добавляет новую оценку для конкретного студента.
- Напишите код, который вычисляет **средний балл** для каждого студента и выводит его.

Решение и использование `.items()`

```
grades = {
 'Alice': [85, 90, 92],
 'Bob': [78, 81],
 'Charlie': [95, 98, 100, 91]
}

# 1. Добавляем новую оценку Бобу
grades['Bob'].append(88)

# 2. Вычисляем средний балл
# Используем .items() для одновременной итерации по ключу (student) и значению (scores)
for student, scores in grades.items():
 if scores: # Проверка, что список оценок не пуст
 average = sum(scores) / len(scores)
 print(f"Средний балл {student}: {average:.2f}")
 else:
 print(f"У {student} нет оценок.")
```

Made with GAMMA

VII. Резюме и Домашнее Задание

Словари и множества – это инструменты, которые значительно повышают читаемость, эффективность и выразительность вашего Python-кода.

Словари (dict)

Для **ассоциативного хранения** данных (ключ-значение). **Быстрый доступ** к данным по осмысленному ключу. Используйте .get() для безопасного доступа и .items() для итерации.

Множества (set)

Для работы с **уникальными элементами** и быстрой **проверкой принадлежности** (in). Незаменимы для операций сравнения коллекций (пересечение, разность).

Домашнее Задание

Частотный Анализ

Напишите программу, которая принимает строку и возвращает словарь, где ключ – символ, а значение – его количество вхождений. **Усложнение:** игнорируйте пробелы и регистр (приведите все к нижнему регистру).

Тройное Пересечение

Даны три списка чисел. Используя множества (set), найдите элементы, которые присутствуют **во всех трех** списках. (Подсказка: используйте оператор пересечения &).

Модель Библиотеки (Творческое)

Смоделируйте простую базу данных книг. Используйте словарь, где ключ – ISBN, а значение – другой словарь с полями: название, автор, статус ("выдана"/"в наличии"). Реализуйте функцию "выдачи" книги (смены статуса).

Made with GAMMA