Python: множества (set)
Методичка на 2 пары по 80 минут

Для группы, которая уже прошла list, tuple, dict, циклы и условия (без функций). Дата: 18 февраля 2026

Цели урока

После двух пар студент должен уметь:

- объяснить, что такое set, где он полезен и чем отличается от list/tuple/dict;
- создавать множества разными способами, понимать почему {} - это dict, а не set;
- использовать базовые методы: add, update, remove, discard, pop, clear;
- делать операции множеств: объединение, пересечение, разность, симметрическая разность;
- решать практические задачи (дедупликация, быстрый поиск, анализ общих/отличающихся элементов) и применять set во вложенных структурах с list/dict/tuple.

1. Что такое set и почему он нужен

Set - это коллекция уникальных элементов. Он особенно полезен, когда нужно быстро ответить на вопрос: «Есть ли элемент в наборе?» или «Какие элементы общие/лишние?»

Ключевые свойства

- Уникальность: одинаковые значения хранятся только один раз.
- Нет индексов: нельзя обратиться как s[0].
- Элементы должны быть хэшируемыми: обычно int/str/tuple (из хэшируемых). Нельзя list/dict/set внутри set.
- Быстрые проверки принадлежности: операция x in s обычно выполняется очень быстро.

Мини-таблица сравнения коллекций

Структура	Порядок	Дубли	Изменяемая	Когда выбирать
list	да	да	да	когда важен порядок/индексы и можно хранить повторы
tuple	да	да	нет	когда нужна фиксированная последовательность (например, ключ dict)
dict	по ключам	ключи уникальны	да	когда нужно хранить пары ключ->значение
set	нет	нет	да	когда нужна уникальность и быстрый поиск/сравнение наборов

2. Создание множеств

2.1 Пустое множество

Важно: {} - это dict. Пустой set создаётся только через set().

```
s = set()
print(s)         # set()
print(type(s))   # <class 'set'>

x = {}
print(type(x))   # <class 'dict'>
```

2.2 Из list/tuple/строки

```
nums = [1, 2, 2, 3, 3, 3]
s = set(nums)
print(s)  # {1, 2, 3} (порядок может отличаться)

t = (10, 10, 20)
print(set(t))  # {10, 20}

word = "banana"
print(set(word))  # {'b', 'a', 'n'}
```

2.3 Литерал множества (не пустой)

```
colors = {"red", "green", "blue"}
print(colors)
```

Частая ошибка: пытаться положить list в set

Так нельзя, потому что list изменяемый (не хэшируемый):

```
# s = {[1, 2, 3]}  # TypeError
# s = set([[1, 2], [3, 4]])  # TypeError
```

Но tuple (из хэшируемых) - можно:

```
points = {(10, 20), (5, 7)}
print(points)
```

3. Базовые методы set

Ниже - самые важные методы, которые понадобятся для практики.

3.1 add и update

```
s = set()
s.add(5)
s.add(5)     # дубль не добавится
print(s)     # {5}

s.update([5, 6, 7])
print(s)     # {5, 6, 7}
```

3.2 remove vs discard

remove(x) выбрасывает ошибку, если элемента нет. discard(x) работает безопасно - без ошибки.

```
s = {1, 2, 3}
s.remove(2)      # ок
# s.remove(10)   # KeyError

s.discard(10)    # ок, без ошибки
print(s)
```

3.3 pop и clear

pop() удаляет и возвращает какой-то элемент (какой именно - не гарантируется).

```
s = {10, 20, 30}
x = s.pop()
print("popped:", x)
print("after:", s)

s.clear()
print(s)  # set()
```

4. Операции множеств (математика)

Это то, ради чего set часто и выбирают: быстро и читаемо сравнивать наборы.

```
a = {1, 2, 3, 4}
b = {3, 4, 5}

print(a | b)  # union: {1,2,3,4,5}
print(a & b)  # intersection: {3,4}
print(a - b)  # difference: {1,2}
print(a ^ b)  # symmetric diff: {1,2,5}
```

Полезные проверки

```
x = {1, 2}
y = {1, 2, 3}

print(x <= y)              # x subset y
print(y >= x)              # y superset x
print(x.isdisjoint({5, 6}))  # True, если нет общих
```

5. Быстродействие (Big O - упрощенно)

Идея: set хорош там, где много проверок «есть ли элемент?» и сравнений наборов.

Операция	set (сред.)	list (сред.)	Почему важно
x in ...	O(1)	O(n)	в list часто нужно перебрать элементы, в set проверка обычно быстрая
add(x)	O(1)	append O(1)	append быстрый, но list не обеспечивает уникальность
remove(x)	O(1)	O(n)	в list сначала ищем элемент, затем удаляем
операции наборов	быстро	ручная логика	union/intersection/diff читаемее и часто быстрее

План на 2 пары по 80 минут

Пара	Время	Содержание
1	0-10	Зачем set, свойства, сравнение с list/tuple/dict
1	10-25	Создание set + типичные ошибки
1	25-45	Методы: add/update/remove/discard/pop/clear, in, len
1	45-65	Операции множеств: | & - ^, subset/superset/disjoint
1	65-80	Мини-практика: 5 коротких задач
2	0-10	Повтор и быстрые вопросы
2	10-35	15 примеров: дедупликация, поиск, фильтрация, анализ текста
2	35-70	Вложенные структуры: dict->set, list+set, индексы для ускорения
2	70-80	Итог и чеклист

6. 15 практических примеров (готовые куски кода)

Пример 1. Удаление дублей (порядок не важен)

```
nums = [5,5,2,2,2,3,3,5,1,1,1]
unique_nums = list(set(nums))
print(unique_nums)
```

Пример 2. Удаление дублей с сохранением порядка

```
nums = [5,5,2,2,2,3,3,5,1,1,1]
seen = set()
result = []

for x in nums:
    if x not in seen:
        seen.add(x)
        result.append(x)

print(result)
```

Пример 3. Быстрая проверка по черному списку

```
blacklist = {"spam", "ads", "scam"}
msg = "buy now ads"
words = msg.split()

blocked = False
for w in words:
    if w in blacklist:
        blocked = True

print("blocked:", blocked)
```

Пример 4. Уникальные буквы (игнорируем пробелы)

```
text = "Hello Python"
letters = set()

for ch in text.lower():
    if ch != " ":
        letters.add(ch)
```

```
print(letters)
print("count:", len(letters))
```

Пример 5. Найти дубликаты в списке

```
nums = [1,2,2,3,4,4,4,5]
seen = set()
dups = set()

for x in nums:
    if x in seen:
        dups.add(x)
    else:
        seen.add(x)

print("duplicates:", dups)
```

Пример 6. Общие элементы двух списков

```
a = [1,2,3,4,5]
b = [4,5,6,7]
common = set(a) & set(b)
print(common)
```

Пример 7. Кто ушел / кто пришел

```
old = ["Ayan", "Dana", "Mira", "Ilyas"]
new = ["Dana", "Mira", "Sasha"]

left = set(old) - set(new)
joined = set(new) - set(old)

print("left:", left)
print("joined:", joined)
```

Пример 8. Уникальные слова + частоты (set + dict)

```
text = "cat dog dog bird cat dog"
words = text.split()

unique_words = set(words)
freq = {}

for w in words:
    freq[w] = freq.get(w, 0) + 1

print("unique:", unique_words)
print("freq:", freq)
```

Пример 9. Фильтрация файлов по расширениям

```
allowed = {"png", "jpg", "webp"}
files = ["a.png", "b.txt", "c.jpg", "d.mp4"]

ok = []
for f in files:
    ext = f.split(".")[-1]
    if ext in allowed:
        ok.append(f)

print(ok)
```

Пример 10. Проверка уникальности логинов

```
logins = ["icdfh", "admin", "icdfh", "user", "admin"]
used = set()

for login in logins:
    if login in used:
        print("duplicate:", login)
    else:
        used.add(login)
```

Пример 11. Удалить стоп-слова

```
stop = {"и", "а", "но", "это"}
words = ["это", "очень", "круто", "и", "полезно"]

filtered = []
for w in words:
    if w not in stop:
        filtered.append(w)

print(filtered)
```

Пример 12. Сравнение навыков (что не хватает)

```
need = {"python", "sql", "git", "react"}
student = {"python", "git"}

missing = need - student
print("missing:", missing)
```

Пример 13. dict -> set (у каждого студента уникальные теги)

```
user_tags = {
    "Ayan": {"python", "sql"},
    "Dana": {"python", "react"},
}

user_tags["Ayan"].add("git")

name = "Mira"
if name not in user_tags:
    user_tags[name] = set()
user_tags[name].add("sql")

print(user_tags)
```

Пример 14. list of dict + set (уникальные города)

```
people = [
    {"name": "Ayan", "city": "Almaty"},
    {"name": "Dana", "city": "Astana"},
    {"name": "Mira", "city": "Almaty"},
]

cities = set()
for p in people:
    cities.add(p["city"])

print(cities)
```

Пример 15. set из tuple (координаты)

```
points = set()
points.add((10, 20))
points.add((10, 20))
```

```
points.add((5, 7))

print(points)
print((5, 7) in points)
```

7. Практика: задачи и подробный разбор

Каждая задача: условие -> решение -> разбор. Решения без функций.

Задача 1. Уникальные элементы и их количество

Условие: Дан список чисел. Выведите множество уникальных чисел и количество уникальных.

Решение:

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

unique_set = set(nums)
print("unique:", unique_set)
print("count:", len(unique_set))
```

Разбор: set(nums) удаляет дубликаты. len(...) показывает сколько уникальных значений осталось.

Задача 2. Удалить дубликаты, сохраняя порядок

Условие: Дан список. Сделайте новый список без дублей, но чтобы порядок первого появления сохранился.

Решение:

```
nums = [5, 5, 2, 2, 2, 3, 3, 5, 1, 1, 1]
seen = set()
result = []

for x in nums:
    if x not in seen:
        seen.add(x)
        result.append(x)

print(result)
```

Разбор: seen хранит «что уже встречали». Если элемент новый - добавляем и в set, и в результат. Так порядок сохраняется, потому что мы идем по исходному списку слева направо.

Задача 3. Найти дубликаты

Условие: Дан список чисел. Найдите, какие значения повторяются (верните set дублей).

Решение:

```
nums = [1, 2, 2, 3, 4, 4, 4, 5]
seen = set()
dups = set()

for x in nums:
    if x in seen:
        dups.add(x)
    else:
        seen.add(x)

print(dups)
```

Разбор: если элемент уже был в seen, значит он дубль - кладем в dups (там тоже уникальность).

Задача 4. Общие элементы двух списков

Условие: Даны 2 списка участников. Найдите общий набор участников.

Решение:

```
group_a = ["Ayan", "Dana", "Mira", "Ilyas"]
group_b = ["Dana", "Sasha", "Mira"]

common = set(group_a) & set(group_b)
print(common)
```

Разбор: превращаем оба списка в множества и берем пересечение (&). В результате остаются только общие элементы.

Задача 5. Кто ушел и кто пришел

Условие: Даны списки участников «до» и «после». Найдите: кто ушел и кто новый.

Решение:

```
old = ["Ayan", "Dana", "Mira", "Ilyas"]
new = ["Dana", "Mira", "Sasha"]

left = set(old) - set(new)
joined = set(new) - set(old)

print("left:", left)
print("joined:", joined)
```

Разбор: разность множеств показывает, какие элементы есть в первом наборе и отсутствуют во втором.

Задача 6. Фильтрация по разрешенным значениям (whitelist)

Условие: Дан список файлов. Оставьте только те, расширение которых разрешено.

Решение:

```
allowed = {"png", "jpg", "webp"}
files = ["a.png", "b.txt", "c.jpg", "d.mp4", "e.webp"]

ok = []
for f in files:
    ext = f.split(".")[-1]
    if ext in allowed:
        ok.append(f)

print(ok)
```

Разбор: allowed - set, поэтому проверка ext in allowed выполняется быстро. split('.')[-1] берет расширение.

Задача 7. Уникальные слова текста (set) + частоты (dict)

Условие: Дана строка текста. Получите: (1) множество уникальных слов, (2) словарь частот.

Решение:

```
text = "cat dog dog bird cat dog"
words = text.split()

unique_words = set(words)
freq = {}

for w in words:
    freq[w] = freq.get(w, 0) + 1

print("unique:", unique_words)
print("freq:", freq)
```

Разбор: set(words) дает уникальные слова. dict freq считает, сколько раз каждое слово встретилось. Здесь set и dict идеально работают вместе: set для «какие существуют», dict для «сколько».

Задача 8. Уникальные города из list[dict]

Условие: Дан список словарей людей (name, city). Соберите множество уникальных городов.

Решение:

```
people = [
    {"name": "Ayan", "city": "Almaty"},
    {"name": "Dana", "city": "Astana"},
    {"name": "Mira", "city": "Almaty"},
    {"name": "Ilyas", "city": "Shymkent"},
]

cities = set()
for p in people:
    cities.add(p["city"])

print(cities)
```

Разбор: создаем пустой set и добавляем значение поля 'city'. Повторы автоматически исчезают.

Задача 9. dict -> set: студент -> уникальные курсы

Условие: Дано: student_courses, где значения - списки курсов (возможны дубли). Сделайте новый dict, где значения - set.

Решение:

```
student_courses = {
    "Ayan": ["Python", "SQL", "Python"],
    "Dana": ["React", "Python", "React"],
}

result = {}

for name, courses in student_courses.items():
    result[name] = set(courses)
```

```
print(result)
```

Разбор: set(courses) убирает дубли внутри курсов каждого студента. Это типичный паттерн «dict -> set» для уникальности.

Задача 10. Проверка: есть ли запрещенные слова

Условие: Дан текст и set запрещенных слов. Выведите True/False, есть ли хоть одно запрещенное слово в тексте.

Решение:

```
bad = {"spam", "scam", "ads"}
text = "hello buy now ads"
words = text.split()

found = False
for w in words:
    if w in bad:
        found = True

print(found)
```

Разбор: мы делаем линейный проход по словам текста, а проверка w in bad быстрая.

Задача 11. Уникальные символы пароля

Условие: Дан пароль. Проверьте, что все символы уникальны (без повторов).

Решение:

```
password = "aB3a!"
chars = set()
ok = True

for ch in password:
    if ch in chars:
        ok = False
    else:
        chars.add(ch)

print(ok)
```

Разбор: если символ уже встречался, значит есть повтор. Этот же шаблон часто используют для поиска дублей.

Задача 12. Совместимость ролей

Условие: Есть набор обязательных ролей и набор ролей пользователя. Выведите, хватает ли ролей и какие отсутствуют.

Решение:

```
required = {"user", "editor"}
user_roles = {"user"}

missing = required - user_roles
print("missing:", missing)
print("ok:", len(missing) == 0)
```

Разбор: разность required - user_roles дает именно то, чего не хватает.

8. Частые ошибки и быстрые проверки

- Путаница: {} - это dict, а не set. Пустой set: set().
- Ожидание порядка: set не предназначен для работы «по индексам».
- Попытка положить list/dict/set внутрь set (они не хэшируемые).
- Использование remove вместо discard, когда элемент может отсутствовать (KeyError).
- Использование pop с ожиданием конкретного элемента - pop возвращает произвольный.

Чеклист перед сдачей практики

- использую set, когда нужна уникальность или быстрый in;
- не использую индексы у set;
- понимаю разницу remove/discard;
- для сравнения наборов использую | & - ^;
- для вложенных структур применяю dict -> set, когда нужна уникальность значений.

Конец методички