День №8
Сегодня 🌞

8.0 Введение в множества

- Понятие множества (set) в Python.
- Основные характеристики множеств: уникальность элементов и неупорядоченность.
- Создание множества с использованием фигурных скобок `{}` или конструктора `set()`.
- Что такое хеширование и как это связано с множествами?
- Почему так сложилось что множества могут содержать только неизменяемые (hashable) элементы, такие как числа, строки и кортежи? Почему нельзя добавлять в множество списки и другие множества.
- Что обрабатывается быстрее? Список, множество, или картеж. Почему?

8.1 Методы множеств

- Полная, сводная таблица ВСЕХ методов множеств в Пайтон с их описанием (включая все методы по добавлению элементов, удалению элементов а так же все методы сравнения, пересечения, проверки вхождений и ВСЕ методы множеств которые есть в пайтон)
- Добавление элементов в множество с помощью метода `add()`.
- Удаление элементов с помощью метода `remove()` или `discard()`.
- Проверка наличия элемента в множестве с использованием оператора `in`.
- Получение длины множества с помощью функции `len()`.
- Применение цикла `for` для перебора элементов множества.
- Применение цикла `while` с условием пока множество существует с удалением одного элемента множества на каждой итерации через `pop`

8.2 Продвинутая работа с множествами

- Подробные примеры кода и детальными описаниями, как работают нижеперечисленные методы.
- Использование метода `update()` для объединения множеств.
- Применение метода `intersection()` для нахождения пересечения множеств.
- Использование метода `difference()` для нахождения разности множеств.
- Методы `union()` и `symmetric_difference()` для выполнения соответственно объединения и симметричной разности множеств.
- Использование метода `isdisjoint()` который проверяет, являются ли два множества непересекающимися.
- Использование метода `issubset()` который проверяет, являются ли два множества непересекающимися.
- Использование метода `issuperset()` который проверяет, является ли одно множество надмножеством для другого множества.
- Метод `clear()` для удаления всех элементов множества.

8.3 Пример работы методов множества на практике

- Практический пример с кодом и подробным описанием
- Сравниваем набор покемонов у Алисы и Никиты
- Покемоны Алисы: Пикачу, Иви, Бульазавр, Чаризард, Сквиртл
- Покемоны Никиты: Варортл, Метапод, Амбреон, Пикачу, Пиджи
- Ищем пересечение (есть у обоих)
- Ищем разность (есть у одного)
- Ищем симметричную разницу (какими ребята могли бы обменятся)

8.0 Введение в множества 📚

2.0 Понятие множества (set) в Python

Множество (set) в Python – это неупорядоченная коллекция уникальных элементов. Множества являются изменяемыми, их можно редактировать, но они не включают изменяемых объектов, таких как списки, другие множества или словари.

3.0 Основные характеристики множеств 🌟

- *Уникальность элементов*: В множестве, каждый элемент присутствует единожды. Это означает, что дубликаты автоматически удаляются при добавлении в множество.
- *Неупорядоченность*: Множества не имеют порядка. Это означает, что порядок элементов может быть случайным и изменяться при редактировании множества.

4.0 Создание множества 🛠

Можно создать множество с помощью фигурных скобок `{}` или конструктора `set()`. Пустое множество создается только с помощью конструктора `set()`, так как пустые фигурные скобки `{}` определяют пустой словарь.

Примеры создания множеств:

```
my_set = {1, 2, 3}
print(my_set) # Вывод: {1, 2, 3}
```

```
empty_set = set()
print(empty_set) # Вывод: set()
```

5.0 Хеширование и множества 🔐

Хеширование - это процесс преобразования данных в уникальное значение (хэш), которое можно использовать для уникальной идентификации элемента. В контексте множеств, хеши нужны для определения уникальности элемента и обеспечения быстрого доступа к нему. Такой подход позволяет выполнять отсутствие дубликатов и операции над множествами с большой скоростью.

6.0 Множества и неизменяемые (hashable) элементы 🚫

Так как множества используют хеширование для идентификации элементов, они могут содержать только неизменяемые (hashable) объекты. Изменяемые объекты, такие как списки и множества, нельзя добавить в множество, потому что их хеш может измениться после добавления. В то же время, числа, строки и кортежи являются неизменяемыми и могут быть элементами множеств.

- Допустимые элементы:

```
valid_set = {1, 'Python', (1, 2, 3)}
print(valid_set) # Вывод: {1, 'Python', (1, 2, 3)}
```

- Недопустимые элементы:

```
invalid_set = {1, 'Python', [1, 2, 3]} # Вызовет ошибку TypeError
```

7.0 Скорость работы: список, множество, кортеж 🧿

Скорость работы операций зависит от структуры данных. В целом, множества обрабатываются быстрее, чем списки и кортежи, особенно при выполнении таких операций, как проверка наличия элемента или объединение двух множеств. Дело в том, что множества используют хеширование для обеспечения быстрого доступа к элементам, в то время как списки и кортежи требуют более длительного поиска.

Для сравнения, рассмотрим проверку наличия элемента в различных структурах данных:

```
my_list = [1, 2, 3, 4, 5]
my_set = {1, 2, 3, 4, 5}
my_tuple = (1, 2, 3, 4, 5)

%timeit 3 in my_list  # Время выполнения будет медленнее
%timeit 3 in my_set   # Время выполнения будет быстрее
%timeit 3 in my_tuple # Время выполнения будет медленнее
```

В этом примере, операция `in` будет выполняться быстрее для множества, чем для списка или кортежа.

8.1 Методы множеств в Python 📚

📑Сводная таблица методов множеств в Python

Метод	Описание
`add()`	Добавляет элемент в множество.
`clear()`	Удаляет все элементы из множества.
`copy()`	Возвращает копию множества.
`difference()`	Возвращает множество, содержащее разность между двумя множествами.
`difference_update()`	Удаляет все элементы из множества, которые присутствуют в другом указанном множестве.
`discard()`	Удаляет указанный элемент из множества, если он присутствует.
`intersection()`	Возвращает множество, содержащее пересечение двух множеств.
`intersection_update()`	Оставляет в множестве только элементы, присутствующие в другом указанном множестве (пересечение множеств).
`isdisjoint()`	Возвращает `True`, если два множества не имеют общих элементов (дизъюнктивны).
`issubset()`	Возвращает `True`, если все элементы множества присутствуют в указанном множестве (является подмножеством).
`issuperset()`	Возвращает `True`, если все элементы указанного множества присутствуют в данном множестве (является надмножеством).
`pop()`	Удаляет и возвращает произвольный элемент из множества. Если множество пусто, возбуждает исключение KeyError.
`remove()`	Удаляет указанный элемент из множества. Возбуждает исключение KeyError, если элемент не найден.
`symmetric_difference()`	Возвращает множество, содержащее симметрическую разность двух множеств (элементы, входящие только в одно из множеств).
`symmetric_difference_update()`	Заменяет текущее множество на симметрическую разность двух множеств.
`union()`	Возвращает множество, содержащее объединение двух множеств.

Метод	Описание
`update()`	Добавляет элементы указанного множества в текущее множество (объединение множеств).

➕ Добавление элементов в множество с помощью метода `add()`

Добавление одного элемента в множество можно сделать с помощью метода `add()`.

Пример:

```
my_set = {1, 2, 3}
my_set.add(4)
print(my_set)  # Output: {1, 2, 3, 4}
```

❌ Удаление элементов с помощью метода `remove()` или `discard()`

Удаление элемента из множества можно сделать двумя способами с помощью методов `remove()` и `discard()`. Разница между ними заключается в том, что `remove()` вызывает исключение KeyError, если элемент не найден, в то время как `discard()` просто ничего не делает.

Пример:

```
my_set = {1, 2, 3, 4}
my_set.remove(4)
print(my_set)  # Output: {1, 2, 3}

my_set.discard(5)
print(my_set)  # Output: {1, 2, 3}
```

🤷‍♂️ Проверка наличия элемента в множестве с использованием оператора `in`

Проверка существования элемента в множестве может быть выполнена с помощью оператора `in`.

Пример:

```
my_set = {1, 2, 3, 4}
print(2 in my_set)  # Output: True
```

🔢 Получение длины множества с помощью функции `len()`

Для определения количества элементов в множестве используйте функцию `len()`.

Пример:

```
my_set = {1, 2, 3, 4}
print(len(my_set))  # Output: 4
```

🔁 Применение цикла `for` для перебора элементов множества

Для итерации по элементам множества используйте цикл `for`.

Пример:

```
my_set = {1, 2, 3}
for number in my_set:
    print(number)

# Output:
#   1
#   2
#   3
```

🔃 Применение цикла `while` с условием, пока множество существует, с удалением одного элемента множества на каждой итерации через `pop`

```
my_set = {1, 2, 3}
while my_set:
    elem = my_set.pop()
    print(elem)

# Output:
#   1
```

```
#   2
#   3
```

🚀 8.2 Продвинутая работа с множествами

📚 2.1 Метод update()

Метод `update()` используется для объединения двух множеств. Функция принимает другое множество (или любой итерируемый объект) и добавляет все его элементы к первому множеству. Если элемент уже присутствует в первом множестве, он не добавляется повторно.

Пример кода:

```
fruits = {'apples', 'oranges', 'bananas'}
more_fruits = {'strawberries', 'grapes', 'bananas'}

fruits.update(more_fruits)
print(fruits)
# Результат: {'apples', 'oranges', 'bananas', 'strawberries', 'grapes'}
```

📚 2.2 Метод intersection()

Метод `intersection()` используется для нахождения пересечения множеств. Он возвращает множество, содержащее элементы, которые присутствуют в обоих исходных множествах.

Пример кода:

```
fruits = {'apples', 'oranges', 'bananas'}
citrus = {'oranges', 'lemons', 'limes'}

common_fruits = fruits.intersection(citrus)
print(common_fruits)
# Результат: {'oranges'}
```

📚 2.3 Метод difference()

Метод `difference()` используется для нахождения разности между двумя множествами. Он возвращает множество, содержащее элементы первого множества, которые отсутствуют во втором множестве.

Пример кода:

```
fruits = {'apples', 'oranges', 'bananas'}
citrus = {'oranges', 'lemons', 'limes'}

non_citrus = fruits.difference(citrus)
print(non_citrus)
# Результат: {'apples', 'bananas'}
```

📚 2.4 Методы union() и symmetric_difference()

Метод `union()` используется для выполнения объединения двух множеств. Он возвращает множество, содержащее все элементы обоих множеств. Схож с `update()`, но возвращает новое множество, не изменяя исходные.

Пример кода:

```
fruits = {'apples', 'oranges', 'bananas'}
more_fruits = {'strawberries', 'grapes', 'bananas'}

all_fruits = fruits.union(more_fruits)
print(all_fruits)
# Результат: {'apples', 'oranges', 'bananas', 'strawberries', 'grapes'}
```

Метод `symmetric_difference()` используется для нахождения симметричной разности между двумя множествами. Возвращает множество, содержащее элементы, которые присутствуют только в одном из исходных множеств, но не в обоих.

Пример кода:

```
fruits = {'apples', 'oranges', 'bananas'}
citrus = {'oranges', 'lemons', 'limes'}

unique_fruits = fruits.symmetric_difference(citrus)
print(unique_fruits)
# Результат: {'apples', 'lemons', 'limes', 'bananas'}
```

🧩 2.5 Метод isdisjoint()

Метод `isdisjoint()` проверяет, являются ли два множества непересекающимися. Возвращает True, если множества не имеют общих элементов, и False в противном случае.

Пример кода:

```
fruits = {'apples', 'oranges', 'bananas'}
citrus = {'oranges', 'lemons', 'limes'}

are_disjoint = fruits.isdisjoint(citrus)
print(are_disjoint)
# Результат: False
```

🧩 2.6 Метод issubset()

Метод `issubset()` проверяет, является ли одно множество подмножеством другого множества. Возвращает True, если все элементы первого множества присутствуют во втором множестве, и False в противном случае.

Пример кода:

```
fruits = {'apples', 'oranges', 'bananas'}
citrus = {'oranges'}

is_subset = citrus.issubset(fruits)
print(is_subset)
# Результат: True
```

🧩 2.7 Метод issuperset()

Метод `issuperset()` проверяет, является ли одно множество надмножеством для другого множества. Возвращает True, если все элементы второго множества присутствуют в первом множестве, и False в противном случае.

Пример кода:

```
fruits = {'apples', 'oranges', 'bananas'}
citrus = {'oranges'}

is_superset = fruits.issuperset(citrus)
print(is_superset)
# Результат: True
```

🗑️ 2.8 Метод clear()

Метод `clear()` используется для удаления всех элементов множества. После выполнения операции множество становится пустым.

Пример кода:

```
fruits = {'apples', 'oranges', 'bananas'}

fruits.clear()
print(fruits)
# Результат: set()
```

8.3 Пример работы методов множества на практике

📝 Основные методы работы с множествами в Python

В данной статье мы рассмотрим работу основных методов множеств на практическом примере. Множествами в Python являются объекты типа `set`, которые предоставляют операции над множествами, такие как:

- Пересечение (intersection)
- Разность (difference)
- Симметричная разница (symmetric_difference)

🎓 Практический пример: сравниваем набор покемонов у Алисы и Никиты

Для этого примера используем следующую ситуацию: у Алисы и Никиты есть свои наборы покемонов, и мы хотим определить, какие покемоны есть у обоих, какие есть только у одного из них, и какими покемонами они могут обменяться.

📋 Алисы и Никиты покемоны

- Покемоны Алисы: Пикачу, Иви, Бульбазавр, Чаризард и Сквиртл.
- Покемоны Никиты: Варортл, Метапод, Амбреон, Пикачу и Пиджи.

📜 Создание множеств с покемонами

Сначала создадим множества с покемонами Алисы и Никиты:

```
pokemons_alice = {'Пикачу', 'Иви', 'Бульбазавр', 'Чаризард', 'Сквиртл'}
pokemons_nikita = {'Варортл', 'Метапод', 'Амбреон', 'Пикачу', 'Пиджи'}
```

🔎 Поиск пересечения множеств (есть у обоих)

Выясним, какие покемоны встречаются у обоих друзей, используя метод `intersection()`:

```
common_pokemons = pokemons_alice.intersection(pokemons_nikita)
print(common_pokemons)
```

Результат:

```
{'Пикачу'}
```

Пикачу – единственный общий покемон для Алисы и Никиты.

🔎 Поиск разности множеств (есть только у одного)

Определим, какие покемоны есть только у Алисы и только у Никиты.

```
unique_pokemons_alice = pokemons_alice.difference(pokemons_nikita)
unique_pokemons_nikita = pokemons_nikita.difference(pokemons_alice)

print('Только у Алисы:', unique_pokemons_alice)
print('Только у Никиты:', unique_pokemons_nikita)
```

Результат:

```
Только у Алисы: {'Иви', 'Сквиртл', 'Чаризард', 'Бульбазавр'}
Только у Никиты: {'Амбреон', 'Пиджи', 'Метапод', 'Варортл'}
```

Итак, у Алисы есть уникальные покемоны Иви, Сквиртл, Чаризард и Бульбазавр, а у Никиты – Амбреон, Пиджи, Метапод и Варортл.

🔎 Поиск симметричной разницы (какими покемонами можно обменяться)

Используем метод `symmetric_difference()` для того, чтобы найти покемоны, которыми Алиса и Никита могли бы обменяться:

```
exchange_pokemons = pokemons_alice.symmetric_difference(pokemons_nikita)
print(exchange_pokemons)
```

Результат:

```
{'Сквиртл', 'Иви', 'Бульбазавр', 'Чаризард', 'Метапод', 'Пиджи', 'Амбреон', 'Варортл'}
```

Алиса и Никита могут обменяться покемонами Сквиртл, Иви, Бульбазавр, Чаризард, Метапод, Пиджи, Амбреон и Варортл.

🎉 Вывод

Мы рассмотрели практический пример работы с методами множеств в Python: `intersection()`, `difference()` и `symmetric_difference()`. Эти операции с множествами позволяют быстро анализировать наборы данных и искать различные отношения между ними. Теперь вы можете применять эти методы в своих задачах!

🏆 5 задачек на Множества

🍽️ 1. Общие блюда меню двух ресторанов

Два ресторана предлагают разные меню. Но есть предположение, что некоторые блюда они предлагают общие. Найдите список общих блюд, используя множества.

```
menu_restaurant1 = ['Пицца', 'Паста', 'Рататуй', 'Салат Цезарь', 'Фруктовый салат', 'Тирамису']
menu_restaurant2 = ['Рисотто', 'Паста', 'Паэлья', 'Салат Цезарь', 'Бурито', 'Тирамису']
```

🎬 2. Общие фильмы актеров

Два актера снялись в нескольких фильмах. Найдите фильмы, в которых они снялись вместе, используя множества.

```
actor1_movies = ['Назад в будущее', 'Замена', 'Матрица', 'День сурка', 'Доброе утро, Вьетнам']
actor2_movies = ['Овощеборцы', 'Замена', 'В поисках Немо', 'Титаник', 'Доброе утро, Вьетнам']
```

📚 3. Общие книги двух читателей

Два читателя прочитали разные книги за год. Найдите книги, которые прочитали оба, используя множества.

```
reader1_books = ['1984', 'Улитка на склоне', 'На западном фронте без перемен', 'Атлант расправил плечи', 'Мастер и Маргарита']
reader2_books = ['Цветы для Элджернона', '1984', 'Улитка на склоне', 'Букварь']
```

🌍 4. Общие языки двух полиглотов

Два человека говорят на нескольких языках. Найдите языки, на которых они могут общаться, используя множества.

```
polyglot1_languages = ['Английский', 'Французский', 'Испанский', 'Немецкий', 'Русский']
polyglot2_languages = ['Английский', 'Китайский', 'Испанский', 'Португальский', 'Русский']
```

🤖 5. Общие навыки для двух вакансий

Компания ищет разработчиков Python. Они опубликовали две вакансии с разными требованиями к навыкам. Вам предстоит найти список навыков, требуемых для обеих вакансий, используя множества.

```
job1_skills = ['Python', 'Django', 'REST API', 'SQL', 'Docker', 'RabbitMQ']
job2_skills = ['Python', 'Flask', 'GraphQL', 'SQL', 'Docker', 'Kubernetes']
```