IT CAREER HUB

Урок 17.

Множества

IT CAREER HUB

Множество

Множество (Set) — это изменяемая структура данных, которая хранит уникальные элементы в неупорядоченном виде.

Множество используется для выполнения различных операций с уникальными элементами, таких как объединение, пересечение и разность. Данные в множестве хранятся с использованием хеширования, что обеспечивает быстрый доступ и проверку наличия элементов.

Особенности множества

- **Хеширование элементов**: При добавлении элемента в множество вычисляется его хеш-код с помощью функции `hash()`. Хеш-код используется для определения позиции элемента в памяти.
- **Уникальность элементов**: Если элемент с таким же хеш-кодом уже существует, то новый элемент не добавляется, так как множество поддерживает только уникальные значения.
- **Неупорядоченность**: Множество не сохраняет порядок добавления элементов.
- **Изменяемость**: Множество можно изменять (добавлять и удалять элементы).

Ограничения

- **Только хешируемые объекты**: Множество может хранить только хешируемые объекты, такие как числа, строки и кортежи.
- **Неупорядоченность**: Порядок элементов не сохраняется, что делает невозможным доступ по индексу.

Создание множества

```python
# Создание множества с помощью фигурных скобок

unique_numbers = {1, 2, 3, 4, 5}

print(unique_numbers)
```

```python
# Создание пустого множества с использованием функции set()
empty_set = set()
print(empty_set)


# Ошибка: {} создаёт пустой словарь, а не множество
empty = {}
print(type(empty))
```

Хеш

Хеш — это числовое значение фиксированной длины, которое вычисляется из хешируемых объектов произвольной длины с помощью специальной хеш-функции. Хеширование используется для быстрого поиска и сравнения данных, а также для проверки целостности информации.

Хешируемые объекты — это типы данных, которые могут быть использованы в качестве ключей в хеш-таблицах или сохранены в структурах данных, использующих хеширование для сохранения и получения объектов.

Чтобы объект считался хешируемым, он должен удовлетворять двум условиям:

1. **Неизменность**: Объект должен быть неизменяемым, то есть его содержимое не может быть изменено после создания. Это необходимо для того, чтобы хеш-код объекта оставался постоянным.
2. **Наличие метода** `__hash__()`: Объект должен иметь метод для вычисления хеш-кода, который возвращает целое число, представляющее хеш-значение.

Функция hash

Python предоставляет встроенную функцию `hash()`, которая возвращает хеш-значение для объектов, поддерживающих хеширование.

Примеры хешируемых объектов

```python
# Хеш-код для строки

print(hash("hello"))

print(hash("hello"))  # Хеш-код одинаковый для одного значения

print(hash('1'))
```

```python
# Хеш-код для числа

print(hash(42))

print(hash(1))

print(hash(1.0))


# Хеш-код для bool

print(hash(True))


# Хеш-код для кортежа

my_tuple = (1, 2, 3)

print(hash(my_tuple))
```

Примеры нехешируемых объектов

```python
# Списки являются изменяемыми объектами, поэтому они не поддерживают хеширование

my_list = [1, 2, 3]

# print(hash(my_list))  # Ошибка: TypeError: unhashable type: 'list'
```

Почему важна хешируемость?

Только хешируемые объекты могут выступать в качестве элементов в множествах, так как их хеш-код используется для быстрого доступа и проверки уникальности.

Применение хеширования

- **Хеш-таблицы**: В Python хеширование используется для реализации таких структур данных, как множества и словари.
- **Проверка целостности данных**: Хеш-функции позволяют убедиться, что данные не были изменены.
- **Алгоритмы и структуры данных**: Используется в алгоритмах поиска и шифрования данных.

Хранение множества в памяти

Множество в Python реализовано с использованием **хеш-таблицы**, что позволяет эффективно хранить уникальные элементы и обеспечивать быстрый доступ к ним. Это достигается благодаря хешированию, которое сопоставляет элементы с уникальными индексами (слотами) в памяти.

Принципы хранения и доступа:

1. **Хеширование элементов**: Каждый элемент множества проходит через хеш-функцию, которая возвращает его хеш-код. Этот хеш-код используется для вычисления индекса, по которому элемент будет храниться в памяти.
2. **Структура хранения**: Множество распределяет элементы по слотам в памяти на основе их хеш-кодов. Каждый слот представляет собой ячейку памяти, в которой хранится ссылка на элемент.
3. **Коллизии**: В случае, если два элемента имеют одинаковый хеш-код и попадают в одну и ту же ячейку хеш-таблицы, возникает коллизия. Python использует один из методов разрешения коллизий, а именно использование цепочек (связных списков). Если возникает коллизия, новый элемент добавляется в цепочку в виде следующего элемента списка.
4. **Неупорядоченность элементов**: Элементы в множестве не имеют определённого порядка, так как они распределяются по памяти в зависимости от их хеш-кодов. Это делает невозможным доступ к элементам по индексу и гарантированную последовательность элементов при итерации.
5. **Поиск**: Для поиска элемента он хешируется для определения индекса слота в хеш-таблице, где может находиться данный элемент.

Пример хранения элементов

```python
my_set = {20, 10, 30, 40}


# Каждый элемент проходит через хеш-функцию
print(hash(20))  # Вывод хеш-кода для элемента 20
print(hash(10))  # Вывод хеш-кода для элемента 10
print(hash(30))  # Вывод хеш-кода для элемента 30
```

```
print(hash(40))  # Вывод хеш-кода для элемента 40


print(my_set)
```

IT CAREER HUB

Преобразование в множество

Функция `set()` используется для преобразования других итерируемых объектов (например, списков, строк, объекта `range`) в множество. Преобразование также удаляет дубликаты из коллекции, позволяя получить только уникальные элементы.

```python
# Преобразование списка

numbers = [1, 2, 2, 3, 4, 4, 5]

unique_numbers = set(numbers)

print(unique_numbers)


# Преобразование строки

text = "hello"

unique_chars = set(text)

print(unique_chars)


# Преобразование объекта range

numbers = set(range(10))

print(numbers)
```

Особенности преобразования

- Порядок элементов в результате преобразования не сохраняется, так как множества неупорядочены.
- В множестве остаются только уникальные элементы.

IT CAREER HUB

Задания для закрепления 1

1. Какие числа будут выведены при выполнении следующего кода?

```python
my_set = {5, 10, 15, 5, 20}
print(my_set)
```

a. {5, 10, 15, 5, 20}
b. {10, 15, 20}
c. {5, 10, 15, 20}
d. Ошибка

Посмотреть ответ

2. Какой результат будет выведен при выполнении следующего кода?

```python
my_list = [1, 2, 3]
my_set = {my_list}
print(my_set)
```

a. {[1, 2, 3]}
b. [1, 2, 3]
c. {1, 2, 3}
d. Ошибка: TypeError

Посмотреть ответ

IT CAREER HUB

Методы добавления и удаления элементов

Множества в Python предоставляют удобные методы для добавления и удаления элементов. Эти методы обеспечивают гибкость и безопасность при работе с уникальными значениями.

Метод	**Описание**	**Синтаксис**
`add()`	Добавляет один элемент в множество.	`my_set.add(element)`
`remove()`	Удаляет указанный элемент. Вызывает ошибку `KeyError`, если элемента нет.	`my_set.remove(element)`
`discard()`	Удаляет указанный элемент. Ошибки не возникает, если элемента нет.	`my_set.discard(element)`
`pop()`	Удаляет и возвращает случайный элемент. Вызывает ошибку `KeyError`, если множество пусто.	`removed_element = my_set.pop()`
`clear()`	Удаляет все элементы из множества, делая его пустым.	`my_set.clear()`

Пример: метод add()

```python
# Метод add()

my_set = {1, 2, 3}

my_set.add(4)

my_set.add(3)
```

```
print(my_set)
```

- Добавляет один элемент в множество.
- Если элемент уже существует, то изменений не происходит.

Пример: метод remove()

```python
# Метод remove()

my_set = {1, 2, 3}

my_set.remove(2)

print(my_set)

# my_set.remove(2)  # Попытка повторного удаления вызовет ошибку KeyError
```

- Удаляет указанный элемент из множества.
- Если элемента нет в множестве, вызывается ошибка KeyError.

Пример: метод discard()

```python
# Метод discard()

my_set = {1, 2, 3}

my_set.discard(2)

my_set.discard(4)  # Ошибки не будет

print(my_set)
```

- Удаляет указанный элемент, если он присутствует в множестве.
- Если элемента нет, ошибки не возникает.

Пример: метод pop()

```python
# Метод pop()

my_set = {1, 2}

removed_element = my_set.pop()

print(removed_element)

print(my_set)  # Оставшиеся элементы

print(my_set.pop())

# print(my_set.pop())  # Вызовет ошибку KeyError, так как элементов больше нет.
```

- Удаляет и возвращает случайный элемент из множества.
- Если множество пусто, вызывается ошибка KeyError.

Метод clear()

- Удаляет все элементы из множества, делая его пустым.
- **Синтаксис**: my_set.clear()

Пример

```python
# Метод clear()

my_set = {1, 2, 3}

my_set.clear()
```

```
print(my_set)
```

Метод copy()

Метод `copy()` используется для создания копии множества.

Синтаксис:

```python
new_set = my_set.copy()
```

Пример

```python
# Метод copy()

original_set = {1, 2, 3}

copied_set = original_set.copy()


# Проверим, что это отдельный объект

copied_set.add(4)

print(copied_set)

print(original_set)
```

IT CAREER HUB

Функции и операторы для множеств

Ко множествами можно применять многие встроенные функции:

1. `len()` — возвращает количество элементов в множестве.

```python
my_set = {1, 2, 3, 4}
print(len(my_set))
```

2. `min()` — возвращает минимальный элемент в множестве.

```python
my_set = {10, 20, 5, 8}
print(min(my_set))
```

3. `max()` — возвращает максимальный элемент в множестве.

```python
my_set = {10, 20, 5, 8}
print(max(my_set))
```

4. `sum()` — возвращает сумму всех элементов в множестве.

```python
my_set = {1, 2, 3, 4}
print(sum(my_set))
```

5. `sorted()` — возвращает отсортированный список элементов множества.

```python
my_set = {3, 1, 4, 2}
print(sorted(my_set))
```

А также операторы вхождения:

1. **Оператор** `in` — проверяет, содержит ли множество определённый элемент.

```python
my_set = {1, 2, 3}
print(2 in my_set)
print(4 in my_set)
```

2. **Оператор** not in — проверяет, отсутствует ли элемент в множестве.

```python
my_set = {1, 2, 3}
print(4 not in my_set)
```

IT CAREER HUB

Задания для закрепления 2

1. Какой результат будет выведен при выполнении следующего кода?

```python
my_set = {1, 2, 3}
my_set.add(3)
print(my_set)
```

a. {1, 2, 3, 3}
b. {1, 2}
c. {1, 2, 3}
d. Ошибка

Посмотреть ответ

2. Какой результат будет выведен при выполнении следующего кода?

```python
my_set = {1, 2, 3}

my_set.remove(0)
```

a. {1, 2}
b. {1, 2, 3}
c. {2, 3}
d. Ошибка `KeyError`

Посмотреть ответ

3. Какой метод используется для удаления элемента из множества без вызова ошибки, если элемента нет?

a. `remove()`
b. `discard()`
c. `delete()`
d. `pop()`

Посмотреть ответ

IT CAREER HUB

Операции над множествами

В Python операции с множествами можно выполнять с помощью операторов и методов. Операции с множествами в Python позволяют эффективно выполнять различные задачи, связанные с обработкой и анализом данных.

Виды операций

Методы, представляющие эти операции, делятся на **изменяющие** и **неизменяющие**. Изменяющие методы модифицируют исходное множество, в то время как неизменяющие создают новое множество и не изменяют оригинал. Кроме того неизменяющие методы можно заменить операторами.

Операция	**Описание**	**Оператор**	**Неизменяющий метод**	**Изменяющий метод**
Объединение	Уникальные элементы из обоих множеств	`|`	`.union()`	`.update()`
Пересечение	Элементы, которые присутствуют в обоих множествах	`&`	`.intersection()`	`.intersection_update()`
Разность	Элементы, которые есть в первом множестве, но отсутствуют во втором	`-`	`.difference()`	`.difference_update()`
Симметрическая разность	Элементы, которые присутствуют только в одном из множеств	`^`	`.symmetric_difference()`	`.symmetric_difference_update()`

Объединение множеств

Объединяет элементы двух множеств, возвращая новое множество с уникальными элементами из обоих множеств.

Оператор: |

Методы:

- Неизменяющий: `.union()` — создаёт новое множество.
- Изменяющий: `.update()` — изменяет исходное множество.

```python
set1 = {1, 2, 3}
set2 = {3, 4, 5}

result1 = set1.union(set2)  # Неизменяющий метод
result2 = set1 | set2  # Использование оператора
print(result1)
print(result2)
print(set1)

set1.update(set2)  # Изменяющий метод
print(set1)
```

Пересечение множеств

Возвращает множество, содержащее элементы, которые присутствуют в обоих множествах.

Оператор: &

Методы:

- Неизменяющий: .intersection()
- Изменяющий: .intersection_update()

```python
set1 = {1, 2, 3}
set2 = {2, 3, 4}
result1 = set1.intersection(set2)  # Неизменяющий метод
result2 = set1 & set2  # Использование оператора
print(result1)
print(result2)
print(set1)

set1.intersection_update(set2)  # Изменяющий метод
print(set1)
```

Разность множеств

Возвращает новое множество, содержащее элементы, которые есть в первом множестве, но отсутствуют во втором.

Оператор: -

Методы:

- Неизменяющий: .difference()
- Изменяющий: .difference_update()

```python
set1 = {1, 2, 3}
set2 = {2, 3, 4}
result1 = set1.difference(set2)  # Неизменяющий метод
result2 = set1 - set2  # Использование оператора
print(result1)
print(result2)
print(set1)

set1.difference_update(set2)  # Изменяющий метод
print(set1)
```

Симметрическая разность

Возвращает новое множество, содержащее элементы, которые присутствуют в одном из множеств, но не в обоих одновременно.

Оператор: ^

Методы:

- Неизменяющий: .symmetric_difference()
- Изменяющий: .symmetric_difference_update()

```python
set1 = {1, 2, 3}
set2 = {2, 3, 4}
result1 = set1.symmetric_difference(set2)  # Неизменяющий метод
result2 = set1 ^ set2  # Использование оператора
print(result1)
print(result2)
```

```
print(set1)

set1.symmetric_difference_update(set2)  # Изменяющий метод
print(set1)
```

IT CAREER HUB

Задания для закрепления 3

1. **Какой результат будет выведен при выполнении следующего кода?**

```python
set1 = {1, 2, 3}

set2 = {3, 4, 5}

result = set1 | set2

print(result)
```

a. {1, 2, 3}
b. {1, 2, 3, 4, 5}
c. {3, 4, 5}
d. Ошибка

Посмотреть ответ

2. **Какой метод используется для изменения множества так, чтобы оно содержало только элементы, присутствующие в обоих множествах?**

a. `.update()`
b. `.intersection()`
c. `.intersection_update()`
d. `.difference_update()`

Посмотреть ответ

3. **Какой результат будет выведен при выполнении следующего кода?**

```python
set1 = {10, 20, 30}

set2 = {20, 30, 40}

result = set1 - set2

print(result)
```

a. {20, 30}
b. {10}
c. {10, 40}
d. {10, -40}

Посмотреть ответ

IT CAREER HUB

Отношения между множествами

Python предоставляет операторы и методы для определения отношений между множествами, таких как подмножества, надмножества и равенство. Эти операции помогают сравнивать множества.

Отношение	Описание	Оператор	Метод
Подмножество	Содержит ли одно множество все элементы другого множества	`<=, <`	`.issubset()`
Надмножество	Содержит ли одно множество другое множество целиком	`>=, >`	`.issuperset()`
Равенство множеств	Содержат ли два множества одни и те же элементы	`==`	Нет метода
Неравенство множеств	Содержат ли два множества разные элементы	`!=`	Нет метода
Отсутствие пересечений	Не имеют ли два множества общих элементов	Нет	`.isdisjoint()`

Пример: подмножество

```python
set1 = {1, 2, 3}
set2 = {1, 2}
set3 = {1, 2, 3}
print(set1 <= set2)
print(set2 <= set1)
print(set2 < set1)
print(set1.issubset(set3))
```

Пример: надмножество

```python
set1 = {1, 2, 3}
set2 = {1, 2}
set3 = {1, 2, 3}
print(set1 >= set2)
print(set2 >= set1)
print(set1 > set3)
print(set1.issuperset(set3))
```

Пример: равенство множеств

```python
set1 = {1, 2, 3}
set2 = {3, 2, 1}
print(set1 == set2)
```

Пример: неравенство множеств

```python
set1 = {1, 2, 3}
set2 = {4, 5, 6}
print(set1 != set2)
```

Пример: отсутствие пересечений

Python

```
set1 = {1, 2}
set2 = {3, 4}
set3 = {2, 3}
print(set1.isdisjoint(set2))
print(set1.isdisjoint(set3))
```

IT CAREER HUB

Цикл по множеству

В Python можно использовать цикл for для перебора элементов множества. Поскольку множество — это неупорядоченная коллекция уникальных элементов, порядок их обхода в цикле не гарантируется и может отличаться при каждом запуске программы.

Пример:

```python
my_set = {10, 20, 30, 40, 50}

for item in my_set:
    print(item)
```

IT CAREER HUB

Задания для закрепления 4

1. Какой результат будет выведен при выполнении следующего кода?

```python
set1 = {1, 2, 3}

set2 = {1, 2}

print(set2.issubset(set1))
```

a. `True`
b. `False`
c. Ошибка

Посмотреть ответ

2. Какой результат будет выведен при выполнении следующего кода?

```python
set1 = {5, 6, 7}

set2 = {8, 9, 10}

print(set1.isdisjoint(set2))
```

a. `True`
b. `False`
c. `{5, 6, 7, 8, 9, 10}`

Посмотреть ответ

3. Какой результат будет выведен при выполнении следующего кода?

```python
set1 = {1, 2, 3}
set2 = {3, 4, 5}
```

```
print(set2 >= set1)
```

a. True
b. False

Посмотреть ответ

IT CAREER HUB

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
1. Результат выполнения кода	Ответ: c
2. Результат выполнения кода	Ответ: c
Задания на закрепление 2	Вернуться к заданиям
1. Результат выполнения кода	Ответ: c
2. Результат выполнения кода	Ответ: d
3. Метод для удаления элемента из множества	Ответ: b
Задания на закрепление 3	Вернуться к заданиям
1. Результат выполнения кода	Ответ: b
2. Метод для изменения множества	Ответ: c
3. Результат выполнения кода	Ответ: b
Задания на закрепление 4	Вернуться к заданиям
1. Результат выполнения кода	Ответ: a
2. Результат выполнения кода	Ответ: b
3. Результат выполнения кода	Ответ: b

IT CAREER HUB

Практическая работа

1. Уникальные символы

Напишите программу, которая создает список из всех уникальных символов строки, за исключением пробелов.

Данные:

```python
text = "hello world"
```

Пример вывода:

```
Уникальные символы: ['l', 'r', 'd', 'h', 'o', 'w', 'e']
```

Решение:

```python
text = "hello world"

unique_chars = set(text)

unique_chars.discard(" ")

unique_chars = list(unique_chars)

print("Уникальные символы:", unique_chars)
```

2. Одинаковые элементы

Напишите программу, которая принимает два списка чисел и выводит список, содержащий элементы без повторений, которые присутствуют в обоих списках.

Данные:

```python
list1 = [1, 2, 3, 4, 5]

list2 = [3, 4, 5, 6, 7]
```

Пример вывода:

```
Элементы в обоих списках: [3, 4, 5]
```

Решение:

```python
list1 = [1, 2, 3, 4, 5]

list2 = [3, 4, 5, 6, 7]

intersection = list(set(list1) & set(list2))

print("Пересечение:", intersection)
```