IT CAREER HUB

Урок 18.

Словари, frozenset

IT CAREER HUB

Словари, frozenset

Set comprehension — это способ создания множества на основе другой последовательности или итерируемого объекта с применением условий и выражений.

Синтаксис полностью аналогичен List comprehension, за исключением фигурных скобок вместо квадратных.

Синтаксис:

```python
new_set = {expression for item in iterable}
```

Пример:

```python
# Создание множества с числами

numbers = [1, 2, 2, 4, 7, 8, 8, 10]

even_numbers_set = {num for num in numbers if num % 2 == 0}

print(even_numbers_set)
```

IT CAREER HUB

Задания для закрепления

1. Какой результат будет выведен при выполнении следующего кода?

```python
words = ["apple", "banana", "cherry", "apple"]

unique_lengths = {len(word) for word in words}

print(unique_lengths)
```

a) {5, 6}

b) [5, 6]

c) {4}

d) {5, 6, 6, 5}

frozenset

frozenset — это неизменяемый аналог обычного множества. В отличие от множества, элементы frozenset не могут быть добавлены, удалены или изменены после создания.

Создание frozenset frozenset создаётся с помощью одноимённой функции, в которую передается итерируемый объект:

```python
immutable_set = frozenset([1, 2, 3, 4, 5])

print(immutable_set)

immutable_from_range = frozenset(range(10))

print(immutable_from_range)
```

Хешируемость frozenset

frozenset является неизменяемым и хешируемым объектом. Это позволяет использовать его в качестве элемента множества, в отличие от обычного множества set, которое не может быть элементом другого множества из-за своей изменяемости.

Пример:

```python
# Создание frozenset

frozen_set1 = frozenset([1, 2, 3])

frozen_set2 = frozenset([4, 5, 6])



# Создание множества, содержащего frozenset

set_of_frozensets = {frozen_set1, frozen_set2}
```

```
print(set_of_frozensets)
```

Сравнение set и frozenset set и frozenset — это оба типа множеств, которые поддерживают уникальные элементы и позволяют выполнять множество операций, таких как объединение, пересечение и разность. Однако между ними есть важные различия, которые влияют на их применение.

Характеристика	**set**	**frozenset**
Изменяемость	Изменяемый: поддерживает методы add(), remove(), discard(), pop(), clear().	Неизменяемый: методы изменения отсутствуют. Поддерживаются только методы, возвращающие новый объект.
Хешируемость	Не является хешируемым, поэтому не может быть элементом других множеств или ключом словаря.	Хешируемый: может использоваться как элемент множества или ключ словаря.

IT CAREER HUB

Задания для закрепления

1. Что из перечисленного верно для frozenset?

a) Это изменяемое множество, поддерживающее добавление элементов.

b) Это неизменяемое множество, элементы которого нельзя изменять после создания.

c) Это хешируемый тип данных.

d) frozenset поддерживает методы add() и remove().

2. Какой результат будет выведен при выполнении следующего кода?

```python
immutable_set = frozenset([1, 2, 3])

new_set = immutable_set.union({4, 5})

print(new_set)
```

a) frozenset({1, 2, 3, 4, 5})

b) {1, 2, 3, 4, 5}

c) {4, 5}

d) Ошибка

IT CAREER HUB

Словарь

Словарь (или ассоциативный массив) — это структура данных, представляющая собой неупорядоченную изменяемую коллекцию пар "ключ-значение".

Создание словарей

Словари в Python представляют собой коллекцию пар "ключ-значение". Каждый ключ в словаре уникален и используется для доступа к соответствующему значению. Словари позволяют хранить и быстро получать доступ к данным благодаря использованию ключей, которые могут быть любыми неизменяемыми объектами, такими как строки, числа или кортежи.

Синтаксис создания словаря:

```python
my_dict = {
    key1: value1,
    key2: value2,
    ...
}
```

key — уникальный объект (ключ), по которому можно получить соответствующее значение.

value — данные (значение), которые хранятся по ключу.

1. Создание словаря с использованием фигурных скобок: Наиболее распространённый способ создания словаря, в котором указываются ключи и соответствующие им значения.

```python
person = {"name": "Alice", "age": 30, "city": "New York"}

print(person)
```

2. Создание пустого словаря: Создать пустой словарь для последующего добавления элементов можно с помощью фигурных скобок или функции `dict()`.

```python
empty_dict = {}

print(empty_dict)



empty_dict = dict()

print(empty_dict)
```

Хранение словаря в памяти

Словари в Python реализованы с использованием хеш-таблиц, что обеспечивает быстрый доступ к значениям по ключу.

Как словарь хранится в памяти:

- **Хеширование ключей**: Когда элемент добавляется в словарь, ключ хешируется с помощью хеш-функции. Хеш-функция вычисляет уникальное (в большинстве случаев) хеш-значение, которое определяет, где именно в памяти будет храниться элемент.
- **Хеширование позволяет словарям быстро находить значение по ключу**: доступ по ключу за постоянное время.
- **Хеш-таблица**: Словарь использует хеш-таблицу, в которой хранятся пары "ключ-значение". Позиции в хеш-таблице определяются хеш-значением ключа.
- **Если несколько ключей имеют одинаковое хеш-значение (коллизии)**, используется метод цепочек для разрешения конфликта.
- **Занимаемая память**: Память, занимаемая словарём, может быть больше, чем просто сумма размеров ключей и значений. Это связано с внутренней

структурой хеш-таблицы и необходимостью оставлять "запас" для вставки новых элементов.
- **Порядок элементов**: Начиная с Python 3.7, словари сохраняют порядок вставки элементов.

Преимущества хранения словаря в памяти:

- **Быстрый доступ**: Словари обеспечивают быстрый доступ к элементам благодаря хешированию ключей.
- **Гибкость**: Словари могут хранить данные любых типов в качестве значений (включая списки, множества, другие словари и т.д.).
- **Уникальные ключи**: Каждый ключ в словаре уникален, что упрощает работу с данными, когда требуется быстрый поиск и проверка существования элементов.

Ограничения:

- **Память**: Из-за внутренней структуры хеш-таблицы словари могут занимать больше памяти по сравнению с другими структурами данных (например, списками).
- **Изменяемость ключей**: Ключи должны быть неизменяемыми и хешируемыми (например, строки, числа, кортежи).

Особенности словарей

- **Ключи должны быть уникальными и хешируемыми**:
    - В словарях каждый ключ должен быть уникальным. Если в словарь добавляется пара с уже существующим ключом, предыдущее значение заменяется новым.
    - Ключи должны быть хешируемыми. Это означает, что в качестве ключей могут выступать только неизменяемые типы данных, такие как строки, числа и кортежи, но не списки или другие изменяемые объекты.
- **Гибкость значений**:
    - В качестве значений в словаре могут использоваться любые объекты Python, включая строки, списки, множества, другие словари и прочие типы данных.
- **Быстрый доступ к значениям**:
    - Словари предоставляют быстрый доступ к значениям по ключу с помощью хеш-таблиц, что делает поиск, добавление и удаление элементов эффективным по времени.
- **Сохранение порядка элементов (начиная с Python 3.7)**:

    - В версиях Python 3.7 и выше порядок вставки элементов в словарь сохраняется. Это означает, что элементы будут перечисляться в том порядке, в котором они были добавлены.
- **Изменяемость значений**:
    - Словари — изменяемые структуры данных. Это означает, что вы можете изменять значения, добавлять и удалять пары "ключ-значение" после создания словаря.

Особенности хеширования

В Python числа `1`, `1.0` и `True` обладают одинаковым хешем:

```python
print(hash(1))

print(hash(1.0))

print(hash(True))
```

Последствия для словарей и множеств

При использовании `1`, `1.0` и `True` в качестве ключей в одном словаре или элементах множества, они будут рассматриваться как один и тот же ключ или элемент. Например:

```python
my_dict = {1.0: "float", 1: "integer", True: "boolean"}

print(my_dict)
```

В этом примере значение `1.0: "float"` будет перезаписано значением `'boolean'`, так как все три ключа считаются одинаковыми. При этом ключ остается первый из добавленных.

Множества также будут содержать только один из таких элементов:

```python
my_set = {True, 1, 1.0}
```

```
print(my_set)
```

IT CAREER HUB

Задания для закрепления

1. Какие из перечисленных типов данных можно использовать в качестве ключа в словаре?

   a) int

   b) bool

   c) set

   d) float

   e) dict

   f) tuple

   g) list

   h) frozenset

2. Какой результат будет выведен при выполнении следующего кода?

```python
my_dict = {1: "one", 1.0: "float one", True: "boolean one"}

print(my_dict)
```

a) {1: "one", 1.0: "float one", True: "boolean one"}

b) {1: "one"}

c) {1: "boolean one"}

d) {1.0: "float one"}

IT CAREER HUB

Преобразование в словарь

В Python можно преобразовать другие коллекции в словарь, если они содержат данные в формате, совместимом со структурой "ключ-значение".

Особенности преобразования:

- Ключи в словаре должны быть хешируемыми и уникальными.
- При преобразовании коллекций с дублирующимися ключами будет сохранено только последнее значение.
- Если структура данных не совместима с форматом "ключ-значение", преобразование в словарь приведёт к ошибке.

1. Преобразование именованных аргументов в словарь: Функция dict() позволяет создавать словарь из пар ключ-значение, передаваемых как именованные аргументы.

Python
```python
# Словарь с использованием функции dict()

person = dict(name="Bob", age=25, city="London")

print(person)
```

2. Преобразование других коллекций в словарь: Функция dict() может быть использована для преобразования последовательности пар (например, списка кортежей) в словарь. Важно чтобы вложенные коллекции состояли ровно из двух элементов и имели неизменяемый элемент на нулевом индексе.

Python
```python
# Список кортежей

pairs = [("name", "Charlie"), ("age", 35), ("city", "Paris")]

person = dict(pairs)

print(person)
```

```python
# Список списков

pairs = [("name", "Charlie"), ["age", 35], ["city", "Paris"]]

person = dict(pairs)

print(person)


# Несоответствующее количество элементов

not_pairs = [("name", "Charlie"), ["age", 35], ["city", "Paris", "Berlin"]]

person = dict(not_pairs)  # Вызовет ошибку

print(person)


# Нехешируемый ключ

pairs = [(["name", "surname"], "Charlie"), ["age", 35], ["city", "Paris"]]

person = dict(pairs)  # Вызовет ошибку

print(person)
```

Доступ к значениям по ключам

Для доступа к значениям в словаре по ключам можно использовать простой синтаксис с квадратными скобками. Это позволяет быстро получать значение, связанное с указанным ключом.

Синтаксис:

```python
variable = dictionary[key]
```

dictionary — объект словаря, из которого вы хотите получить значение.

key — ключ, по которому будет извлечено связанное значение.

variable — переменная, в которой можно сохранить значение, связанное с указанным ключом.

Особенности:

- Если ключ существует, будет возвращено соответствующее значение.
- Если ключ отсутствует, возникает ошибка KeyError.

Примеры:

```python
# Получение значения по ключу

my_dict = {"name": "Alice", "age": 30}

print(my_dict["name"])

age = my_dict["age"]

print(age)

# print(my_dict["city"])  # Вызовет ошибку
```

IT CAREER HUB

Задания для закрепления

1. Какой результат будет выведен при выполнении следующего кода?

```python
pairs = [("name", "Charlie"), ("age", 35), ("city", "Paris"), ("name", "Bob")]

person = dict(pairs)

print(person["name"])
```

a) Charlie
b) ("name", "Charlie")
c) "name": Charlie
d) Bob

2. Какой результат будет выведен при выполнении следующего кода?

```python
not_pairs = [("name", "Charlie"), ["age", 35], ["city", "Paris", "Berlin"]]

person = dict(not_pairs)

print(person)
```

a) {"name": "Charlie", "age": 35, "city": "Paris"}

b) {"name": "Charlie", "age": 35, "city": ["Paris", "Berlin"]}

c) {"name": "Charlie", "age": 35, "city": "Berlin"}

d) Ошибка

Оператор in

Оператор in используется для проверки, существует ли указанный ключ в словаре.

Он часто используется для проверки наличия ключа перед доступом к значению, чтобы избежать ошибки KeyError.

Пример:

```python
my_dict = {"name": "Alice", "age": 30}

if "name" in my_dict:

    print(my_dict["name"])  # Выведет значение по ключу 'name'

if "city" in my_dict:

    print(my_dict["city"])  # Не выполняется, так как ключ 'city' отсутствует
```

Цикл по словарю

По словарям можно итерироваться с помощью цикла for. По умолчанию цикл for перебирает ключи словаря, с помощью которых можно получить значения.

```python
my_dict = {"name": "Alice", "age": 30, "city": "New York"}

for key in my_dict:

    print(f"Ключ={key}, значение={my_dict[key]}")
```

Добавление и обновление данных

Словари позволяют быстро добавлять новые пары "ключ-значение" или обновлять существующие значения по ключу.

Добавление новой пары "ключ-значение"

```python
my_dict = {"name": "Alice", "age": 30}

my_dict["city"] = "New York"  # Добавление нового элемента

print(my_dict)
```

Обновление значения по существующему ключу

```python
my_dict = {"name": "Alice", "age": 30}

my_dict["age"] = 31  # Обновление значения по ключу "age"

print(my_dict)
```

Метод update()

Метод update() позволяет добавлять несколько пар "ключ-значение" или обновлять значения существующих ключей.

Можно передать другой словарь или другую последовательность пар ключ-значение.

```python
# Обновление значений и добавление новых ключей

my_dict = {"name": "Alice", "age": 30}

my_dict.update({"age": 32, "country": "USA"})

print(my_dict)



# Обновление с использованием списка кортежей

my_dict.update([("name", "Bob"), ("email", "bob@example.com")])

print(my_dict)
```

```
# Обновление с использованием именованных аргументов

my_dict.update(city="New York", orders=[])

print(my_dict)
```

Особенности добавления и обновления: Если ключ уже существует, его значение будет обновлено. Если ключ отсутствует, он будет добавлен в словарь.

Удаление данных

Оператор del удаляет элемент с указанным ключом из словаря. Если ключ отсутствует, возникает ошибка KeyError.

Python
```
my_dict = {"name": "Alice", "age": 30, "city": "New York"}

del my_dict["age"]

print(my_dict)

# del my_dict["email"]  # Вызовет ошибку
```

Метод clear() удаляет все элементы из словаря, оставляя пустой словарь.

Python
```
my_dict = {"name": "Alice", "age": 30}

my_dict.clear()

print(my_dict)
```

Удаление и получение данных

Метод pop() - удаляет элемент по указанному ключу и возвращает его значение. Если ключ отсутствует и значение по умолчанию не указано, возникает ошибка KeyError.

Python

```python
my_dict = {"name": "Alice", "age": 30}

age = my_dict.pop("age")

print(age)

print(my_dict)

# my_dict.pop("email")  # Вызовет ошибку
```

Метод popitem() - удаляет и возвращает последнюю добавленную пару (начиная с Python 3.7) или случайную пару. Если словарь пуст, возникает ошибка KeyError.

Python

```python
my_dict = {"name": "Alice", "age": 30}

last_item = my_dict.popitem()

print(last_item)

print(my_dict)
```

IT CAREER HUB

Задания для закрепления

1. Какой результат будет выведен при выполнении следующего кода?

```python
my_dict = {"name": "Alice", "age": 30}

my_dict.update({"city": "New York", "age": 35})

print(my_dict)
```

a) {"name": "Alice", "age": 30, "city": "New York"}
b) {"name": "Alice", "age": 35, "city": "New York"}
c) {"name": "Alice", "age": 30, "age": 35, "city": "New York"}
d) Ошибка

2. Какой результат будет выведен при выполнении следующего кода?

```python
my_dict = {"name": "Alice", "age": 30}

del my_dict["age"]

print(my_dict)
```

a) {"name": "Alice"}
b) {"name": "Alice", "age": 30}
c) {"name": "Alice", "age"}
d) {"name": "Alice", 30}

3. Какое значение будет возвращено при выполнении следующего кода?

```python
my_dict = {"name": "Alice", "age": 30}
```

```
value = my_dict.pop("age")

print(value)
```

a) {"age": 30}
b) {"name": "Alice"}
c) 30
d) None

IT CAREER HUB

Практические задания

Инверсия словаря

Напишите программу, которая создаёт новый словарь, где значения из исходного словаря становятся ключами, а ключи — значениями.

Данные:

```python
original_dict = {"a": 1, "b": 2, "c": 3}
```

Пример вывода:

```python
Инверсированный словарь: {1: 'a', 2: 'b', 3: 'c'}
```

Решение:

```python
original_dict = {"a": 1, "b": 2, "c": 3}


inverted_dict = {}


for key in original_dict:

    inverted_dict[original_dict[key]] = key


print("Инверсированный словарь:", inverted_dict)
```

Из чисел в слова

Напишите программу, которая заменяет числовые значения в словаре на их строковое представление (например, 1 → "один"). Используйте заранее подготовленный словарь чисел.

Данные:

```python
# Словарь сопоставлений

number_to_word = {1: "один", 2: "два", 3: "три"}

# Исходные данные

data = {"x": 1, "y": 2, "z": 3}
```

Пример вывода:

```python
{'x': 'один', 'y': 'два', 'z': 'три'}
```

Решение:

```python
data = {"x": 1, "y": 2, "z": 3}

number_to_word = {1: "один", 2: "два", 3: "три"}



for key in data:

    if data[key] in number_to_word:

        data[key] = number_to_word[data[key]]



print(data)
```

IT CAREER HUB