Сообщить об ошибке.

Обзор возможностей словаря dict в Python

Справочник по языку Python3. / Обзор возможностей словаря dict в Python

Что такое словарь Python и как его применять

Словари встречаются и в других языках как, только называются по разному, например "ассоциативная память" или "ассоциативные массивы". В отличие от последовательностей, которые индексируются диапазоном чисел, словари индексируются ключами, которые могут быть любого неизменяемого типа. Строки и числа всегда могут быть ключами. Кортежи могут использоваться в качестве ключей, если они содержат только строки, числа или кортежи. Если кортеж содержит любой изменяемый объект прямо или косвенно, он не может использоваться в качестве ключа.

Нельзя использовать списки в качестве ключей словаря, так как списки могут быть изменены на месте с помощью индексов, срезов или методов, таких как append() и extend().

Лучше всего рассматривать словарь как набор пар "ключ-значение" с требованием, чтобы ключи были уникальными в пределах одного словаря. Пара фигурных скобок создает пустой словарь: '{}'. Размещение разделенного запятыми списка пар key: value в фигурных скобках добавляет пары key: value в словарь. Так же словари записываются в коде.

Основное применение словаря - это хранение значения с некоторым ключом и извлечение значения из словаря, заданного ключом. Также можно удалить пару key:value используя инструкцию del.

Если в словарь добавляется новый ключ, который уже используется, старое значение, связанное с этим ключом, будет забыто. Извлечение значения с помощью несуществующего ключа является ошибкой. Если ошибки при извлечении значений не желательны, то необходимо использовать метод словаря dict.get(). Метод dict.get() возвращает None или указанное значение по умолчанию, если ключа не существует.

Выполнение list(d) в словаре возвращает список всех ключей, используемых в словаре, в порядке вставки. Если вы хотите, чтобы он был отсортирован, просто используйте sorted(d).

Варианты использования словарей dict в Python.

- Основные операции со словарями;
- Варианты создания словаря;
- Списки-представления словаря;
- Выражение-генератор словаря;
- Использование функций в качестве значений словаря;
- Имитация конструкции switch/case словарем Python.

Основные операции со словарями Python.

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
# {'jack': 4098, 'sape': 4139, 'guido': 4127}
```

```
# извлечение значения по ключу
>>> tel['jack']
# 4098

# извлечение значения по ключу со значением
# по умолчанию (если такого ключа нет)
>>> tel.get('joy', 2021)
# 2021

# удаление и добавление элементов словаря
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
# {'jack': 4098, 'guido': 4127, 'irv': 4127}

# преобразование ключей словаря в список
>>> list(tel)
# ['jack', 'guido', 'irv']

# сортировка словаря
>>> sorted(tel)
# ['guido', 'irv', 'jack']

# проверка наличия ключа в словаре
>>> 'guido' in tel
# True

>>> 'jack' not in tel
# False
```

Варианты создания словаря Python.

Класс dict() строит словарь непосредственно из последовательностей пар ключ-значение:

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
# {'sape': 4139, 'guido': 4127, 'jack': 4098}
```

Когда ключи являются строками, то проще указать пары, используя ключевые аргументы:

```
>>> dict(sape=4139, guido=4127, jack=4098)
# {'sape': 4139, 'guido': 4127, 'jack': 4098}
```

Для создания словарей из произвольных ключей и значений можно использовать генераторы-словарей.

```
>>> {x: x**2 for x in (2, 4, 6)}
# {2: 4, 4: 16, 6: 36}

# словарь генерируется из 2-х списков
>>> {x: y for x, y in zip(['a', 'b', 'c'], [1, 2, 3])}
# {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}
```

Списки-представления словаря Python.

Словарь содержит очень полезные методы, которые называются списки-представления dict.keys(), dict.values(), dict.items(), которые изменяются динамически. Это значит, что все изменения, такие как удаление, изменение или добавление значений в словаре сразу отражаются на соответствующем представлении.

```
>>> x = {'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> keys = x.keys()
>>> values = x.values()
>>> items = x.items()

# Производим операции со словарем 'x', а все
# отражается на списках-представлениях
>>> x['one'] = 0
>>> values
# dict_values([0, 2, 3, 4])
>>> items
# dict_items([('one', 0), ('two', 2), ('three', 3), ('four', 4)])
```

```
>>> x['ten'] = 10
>>> keys
# dict_keys(['one', 'two', 'three', 'four', 'ten'])
>>> values
# dict_values([0, 2, 3, 4, 10])
>>> items
# dict_items([('one', 0), ('two', 2), ('three', 3), ('four', 4), ('ten', 10)])

>>> del x['three']
>>> items
# dict_items([('one', 0), ('two', 2), ('four', 4), ('ten', 10)])
>>> values
# dict_values([0, 2, 4, 10])
>>> keys
# dict_keys(['one', 'two', 'four', 'ten'])
```

Обратите внимание, что присвоение значений keys, values и items происходит только в начале кода, а отслеживание изменений в словаре идет до конца кода.

С помощью списков-представлений можно проверить наличие в словаре определенного **значения** ключа или проверить наличие пары "ключ-значение", для этого используется оператор проверки вхождения in.

```
>>> x = {'one': 0, 'two': 20, 'three': 3, 'four': 4}
>>> keys = x.keys()
>>> 'one' in keys
# True

>>> values = x.values()
>>> 3 in values
# True

>>> items = x.items()
>>> ('three', 3) in items
# True
```

Выражение-генератор словаря Python.

При помощи выражения генератора словаря можно сделать много интересного.

Пример обмена местами (инверсии) ключей и значений словаря, при этом помним, что ключом может быть только неизменяемый объект.

```
>>> d = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
>>> {y: x for x, y in d.items()}
# {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}
```

Пример фильтра ключей и значений словаря. Отберем ключи и значения словаря, отвечающие определенным условиям. В результате будет создан новый словарь.

```
>>> d = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
# отберем элементы словаря, ключи которых имеют значения 'a' или
# 'c' или 'e', а значения этих ключей должны быть больше 1
>>> {key: val for key, val in d.items() if key in ('a', 'c', 'e') and val > 1}
# {'c': 3, 'e': 5}
```

Пример сортировки словаря по значению:

```
>>> x = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
>>> {k: v for k, v in sorted(x.items(), key=lambda item: item[1])}
# {0: 0, 2: 1, 1: 2, 4: 3, 3: 4}
```

или так:

```
>>> x = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
>>> dict(sorted(x.items(), key=lambda item: item[1]))
# {0: 0, 2: 1, 1: 2, 4: 3, 3: 4}
```

Использование функций в качестве значений словаря Python.

Так как функции в Python являются объектами, то можно заменить значения ключей словаря функциями и возвращать их, если ключи совпадают.

Очень простой пример.

```
# словарь с функциями
calc = {
    "plus": lambda x, y: x + y,
    "minus": lambda x, y: x - y,
    "division": lambda x, y: x / y,
    # в качестве значения используем
    # встроенную функцию pow()
    "power": pow
}

# типа фабрики функций
def action(match, dictionary, default="NO CALC"):
    for key in dictionary.keys():
        if match in key:
            return dictionary[key]
    return lambda *x: default

>>> plus = action('plus', calc)
>>> minus = action('minus', calc)
>>> power = action('power', calc)
>>> square = action('square', calc)
>>> plus(5, 4)
# 9
>>> minus(5, 4)
# 1
>>> power(3, 3)
# 27
>>> square(1, 1)
'NO CALC'
>>> square(1)
'NO CALC'
```

Можно использовать словарь с функциями напрямую, если лень создавать фабрику (далее будет рассмотрена фабрика на collections.defaultdict() и лямбда-функциях):

```
# словарь с функциями
calc = {
    "plus": lambda x, y: x + y,
    "minus": lambda x, y: x - y,
    "division": lambda x, y: x / y,
    # в качестве значения используем
    # встроенную функцию pow()
    "power": pow
}

>>> calc['plus'](5, 4)
# 9
>>> calc['minus'](5, 4)
# 1
>>> calc['division'](9, 3)
# 3.0
>>> calc['power'](3, 3)
# 27
```

Передача аргументов функциям, расположенным в словаре очень проста. Однако что, если нужно манипулировать аргументами перед передачей их в функцию? В этом случае код может выглядеть так:

```
def handle_event(e):
    print(f"Обработка события в 'handler_event' с помощью: {e}")
    return e

def handle_other_event(e):
```

```
    print(f"Обработка события в 'handle_other_event' с помощью: {e}")
    return e

functions = {
    "event1": lambda arg: handle_event(arg["some-key"]),
    "event2": lambda arg: handle_other_event(arg["some-other-key"]),
}

event = {
    "some-key": "value",
    "some-other-key": "different value",
}

print(functions["event1"](event))
# Обработка события в 'handler_event' с помощью: value
# value
print(functions["event2"](event))
# Обработка события в 'handle_other_event' с помощью: different value
# different value
```

Имитация конструкции switch/case словарем Python.

В случае, если необходимо эмулировать поведение операторов switch/case, то следует рассмотреть возможность использования значения по умолчанию, когда ключ словаря отсутствует. Плюс ко всему здесь будем использовать диспетчеризацию для функций, требующих более одной строки кода (лямбда-функции хороши для простых случаев).

Вот как это сделать:

```
from collections import defaultdict

def add(x, y):
    return x + y

def mul(x, y):
    return x * y

cases = defaultdict(lambda *args: lambda *a: "Invalid option", {
    "add": add,
    "mul": mul,
})

>>> cases["add"](5, 3))
# 8
>>> cases["plus"](5, 3))
# Invalid option
```

Единственная разница при использовании правильных функций заключается в том, что они должны быть определены вне словаря, потому что Python не допускает встроенных определений функций.

Этот фрагмент использует collections.defaultdict, первый аргумент которого указывает "*фабрику по умолчанию*", которая представляет собой функцию, которая будет вызываться, когда ключ не найден. Можно заметить, что здесь используются две лямбда-функции. Первая предназначена для перехвата любого количества переданных ему аргументов, а вторая - для того чтобы вернуть вызываемый объект.

> *Дополнительно смотрите:*
>
> - *Встроенный класс dict()*.
> - *Тип dict, словари в Python*.
> - *Методы словарей и поддерживаемые операции*.