eebb

ЯЗЫК PYTHON

Лекция 10. Словари

Словари

- В чем-то похожи на множества, в чем-то на списки
- Как будто к элементу множества привязано значение
- Как будто список, только индекс - не обязательно число
- А любой неизменяемый тип
- Часто - строка

Примеры словарей

- Данные о пользователе по идентификатору
- Или о любом объекте
- Счетчик слов в тексте
- Результаты онлайн-тестирования (баллы по пользователю)
- Результаты обхода web-ресурсов (содержимое по URL)

Словарь как литерал

- Фигурные скобки
- В скобках - пары ключ-значение
- Ключ отделен от значения двоеточием
- Пары отделены друг от друга запятой
- Пустые фигурные скобки - пустой словарь

Пример

```
1 name_by_id = {'id-12345': 'vasya', 'id-23456': 'petya'}
2 print(name_by_id)
3 empty = {}
4 print(empty)
5 name_by_id = {
6     'id-12345': 'vasya',
7     'id-23456': 'petya',
8 }
9 print(name_by_id)
```

Добавление элементов

- Как присваивание элемента в списке
- В квадратных скобках - ключ
- Справа - новое значение
- Создает новую пару или меняет значение ключа

Пример

```
1 name_by_id = {}
2 name_by_id['id-12345'] = 'vasya'
3 print(name_by_id)
4
5 name_by_id['id-23456'] = 'petya'
6 print(name_by_id)
7
8 name_by_id['id-12345'] = 'dima'
9 print(name_by_id)
```

Получение элемента по ключу

- Два способа
- Первый - по аналогии со списком
- Ключ в квадратных скобках
- Если ключа нет - бросается исключение

Пример

```
 1 name_by_id = {}
 2 name_by_id['id-12345'] = 'vasya'
 3 print(name_by_id['id-12345'])
 4 name_by_id['id-23456'] = 'petya'
 5 print(name_by_id['id-12345'])
 6 print(name_by_id['id-23456'])
 7 name_by_id['id-12345'] = 'dima'
 8 print(name_by_id['id-12345'])
 9 print(name_by_id['id-23456'])
10 print(name_by_id['id-34567']) # здесь будет исключение
```

Получение элемента по ключу

- Второй - метод get
- С одним параметром или с двумя
- Первый параметр - по-любому ключ
- Если второй параметр есть - это значение по умолчанию
- Если нет - по умолчанию возвращаем None

Пример

```
1 name_by_id = {}
2 print(name_by_id.get('id-12345')) # None
3 print(name_by_id.get('id-12345', '')) # ''
4 name_by_id['id-12345'] = 'vasya'
5 print(name_by_id.get('id-12345')) # 'vasya'
6 print(name_by_id.get('id-12345', '')) # 'vasya'
7 print(name_by_id.get('id-23456')) # None
8 print(name_by_id.get('id-123456', '')) # ''
```

Цикл `for`

- Словарь можно указать справа от in
- Это будет цикл по ключам
- В порядке их добавления
- Изменение значения по существующему ключу не влияет на порядок перебора
- Изменение словаря во время такого цикла может привести к исключению

Пример

```
name_by_id = {}
for k in name_by_id:
    print(k)

name_by_id['id-12345'] = 'vasya'
for k in name_by_id:
    print(k)

name_by_id['id-23456'] = 'petya'
for k in name_by_id:
    print(k)


name_by_id['id-12345'] = 'dima'
for k in name_by_id:
```

Удаление ключа

- Есть команда del
- В общем случае она удаляет переменную (del v)
- Можно указать словарь с квадратными скобками и ключом
- Это удалит ключ из словаря
- При отсутствии ключа - исключение

Пример

```
1 name_by_id = {}
2 name_by_id['id-12345'] = 'vasya'
3 name_by_id['id-23456'] = 'petya'
4 del name_by_id['id-12345']
5 print(name_by_id)
6 del name_by_id['id-12345']  # исключение
```

Удаление ключа и порядок обхода

- Удаление влияет на порядок обхода
- Если ключ удален и потом добавлен
- То он считается добавленным заново
- Иногда это важно

Пример

```
name_by_id = {}
for k in name_by_id:
    print(k)
print()

name_by_id['id-12345'] = 'vasya'
for k in name_by_id:
    print(k)
print()

name_by_id['id-23456'] = 'petya'
for k in name_by_id:
    print(k)
print()

```

Функция-конструктор

- dict - функция-конструктор словаря
- Без параметров - пустой словарь
- Параметр-словарь - порождает копию
- Параметр-коллекция - работает только для коллекций пар

Функция-конструктор

- Пары - не обязательно кортежи
- Пойдет любая упорядоченная
- Эффект - как будто прошли по коллекции пар
- И добавили по ключу-первому элементу значение-второй

Пример

```
 1 d = dict()
 2 print(d)
 3
 4 d = dict({'id1': 'name1', 'id2': 'name2'})
 5 print(d)
 6
 7 d = dict([('id1', 'name1'), ('id2', 'name2')])
 8 print(d)
 9
10 d = dict(('ab', 'cd', 'ef'))
11 print(d)
```

Словарь в других функциях-конструкторах

- Логика аналогична циклу for
- Словарь сводится к набору ключей
- list вернет список ключей, tuple - кортеж ключей
- str стоит особняком
- Он вернет строковое представление

Пример

```
1 d = {'id1': 'name1', 'id2': 'name2'}
2 print(list(d))
3 print(tuple(d))
4 print(set(d))
5 print(str(d))
```

Другие переборы

- Иногда хочется перебрать словарь не как набор ключей
- Для этого есть методы
- items - возвращает последовательность пар ключ-значение
- values - возвращает последовательность значений
- keys - возвращает последовательность ключей

Пример

```
1 d = {'id1': 'name1', 'id2': 'name2'}
2 d_rev = {}
3
4 for k, v in d.items():
5     d_rev[v] = k
6
7 print(d_rev)
```

Изменяемость

- Принадлежность пары к словарю определяется ключом
- Значение - просто "подвешивается" к ключу
- Поэтому ключ обязан быть неизменяемым
- Значение - может быть изменяемым

Пример

```
1 d = {'id1': 'name1', 'id2': 'name2'}
2 d['key'] = 'value'
3 d['key'] = ['list', 'value']
4 d[['list', 'key']] = 'value' # нельзя
```

Вернемся к функциям

- Мотивирующий пример
- Есть функция для печати данных о пользователе
- Среди них есть обязательные атрибуты
- Например, имя, дата рождения

Вернемся к функциям

- А есть атрибуты, свойственные не всем
- Например, год окончания вуза
- А есть такие, которые появляются по ходу
- Про которые мы могли не знать на момент написания функции

Вернемся к функциям

- Хотим написать функцию один раз
- Чтобы она принимала набор стандартных атрибутов пользователя
- И их заранее известным образом обрабатывала и печатала
- А в конце печатала все дополнительные атрибуты

Вернемся к функциям

- Это можно решить и стандартными средствами
- Например, завести именованные параметры для стандартных атрибутов
- И отдельный параметр other для других
- Передавать его словарем

Вернемся к функциям

- Придется создавать словарь
- Только для передачи этих атрибутов
- Это будет некрасиво и нелогично
- Одни атрибуты передаем отдельными параметрами
- А остальные - словарем

Произвольные ключевые параметры

- В определении функции один из параметров можем пометить двойной звездочкой
- В него будут собраны все неизвестные именованные параметры
- Для тела функции это будет словарь
- Ключами будут имена параметров
- Значениями - их значения

Произвольные ключевые параметры

- У функции могут быть именованные параметры
- Они в словарь не попадут
- Такой параметр можно назвать как угодно
- Есть традиция называть его kwds

Пример

```
 1 def print_user(name, address, **kwds):
 2     print("User name:", name)
 3     if address is not None:
 4         print("Address:", address)
 5     else:
 6         print("Address is absent")
 7     for k, v in kwds.items():
 8         print(k + ":", v)
 9
10 print_user('vasya', 'moscow')
11 print_user('vasya', None, age=23, salary=11111)
```

Противоположная задача

- Иногда хочется сделать обратное
- Есть функция с именованными параметрами
- У нас есть ее параметры в виде словаря
- Или часть параметров

Противоположная задача

- Например, мы читаем входной файл с данными о каких-то объектах
- Объекты могут быть разные
- И атрибуты разные
- Хранятся они в текстовом файле

Противоположная задача

- В начале строки имя атрибута
- Потом двоеточие и значение
- Тип объекта представлен так же
- И есть общая функция, которая читает такой файл и возвращает словарь

Противоположная задача

- Мы вызываем такую функцию
- Проверяем тип объекта
- Если это user, то вызываем print_user
- Если какой-то другой - какую-то другую

Противоположная задача

- Можно вставить словарь в список параметров
- И двойную звездочку спереди
- Синтаксис такой же
- Но контекст другой
- И смысл противоположный

Пример

```
 1 def print_user(name, address, **kwds):
 2     print("User name:", name)
 3     if address is not None:
 4         print("Address:", address)
 5     else:
 6         print("Address is absent")
 7     for k, v in kwds.items():
 8         print(k + ":", v)
 9
10
11 data = {'name': 'vasya', 'address': 'qwerty', 'hobby': 'pyt
12 print_user(**data)
```

Для позиционных параметров

- Есть похожие конструкции
- Только одна звездочка вместо двух
- И традиционное имя для параметра - args
- В нее собираются все позиционные параметры
- И можно вставить содержимое списка в параметры в точке вызова

Замена функции

- Предположим, что есть функция
- И мы хотим видеть все факты ее вызова
- Например, печатать на экран
- И пусть мы хотим делать это для разных функций
- Напишем функцию для преобразования функции
- Будем считать, что преобразуем функцию с одним параметром

Пример

```
 1 def print_call(f):
 2     def worker(p):
 3         print("call function with ", p)
 4         return f(p)
 5 
 6     return worker
 7 
 8 def incr(v):
 9     return v + 1
10 
11 incr = print_call(incr)
12 
13 print(incr(5))
```

Обобщим

- Все хорошо, но обрабатываем частный случай
- Хотелось бы общий вариант
- Тут помогает обобщенная работа с аргументами
- Обоих типов и в обе стороны

Пример

```
 1 def print_call(f):
 2     def worker(*args, **kwds):
 3         print("call function with ", args, kwds)
 4         return f(*args, **kwds)
 5 
 6     return worker
 7 
 8 def incr(v):
 9     return v + 1
10 
11 incr = print_call(incr)
12 
13 print(incr(5))
```

Пример

```
1 def sum(a, b):
2     return a + b
3
4 sum = print_call(sum)
5
6 print(sum(5, 10))
7 print(sum(a=5, b=10))
```

Декораторы

- В Python есть специальная конструкция для таких модификаций функций
- Называется декораторы
- Детали не входят в наш курс
- Но идейно это преобразования функций
- Желающим - на самостоятельное изучение

Другие коллекции

- Counter - счетчик по ключам
- frozenset - неизменяемое множество
- deque - очередь, стек
- heapq - очередь с приоритетами

Другие коллекции

- namedtuple - именованные кортежи
- Содержит черты кортежа и словаря
- Можно создавать кортежи, у чьих полей будут имена
- Можно обращаться к элементам по номеру и по имени