PYTHON

Лекция 8

ПЛАН ЛЕКЦИИ

- Словари - остаток
- Множества
- Другие коллекции

KEYS/VALUES/ITEMS

- keys() - метод, возвращающий ключи как view
- Нужен не так часто - в "списочном" контексте и так их получим
- values() - значения как view
- items() - пары ключ-значение как view

ПРИМЕР

```
 1 data = {'vasya': 123, 'petya',: 234}
 2 k = data.keys()
 3 v = data.values()
 4 it = data.items()
 5 list_k = list(data.keys())
 6 list_v = list(data.values())
 7 list_it = list(data.items())
 8
 9 data['dima'] = 345
10 print(k, v, it)
11 print(list_k, list_v, list_it)
```

УДАЛЕНИЕ КЛЮЧА

- Метода remove нет
- Есть pop с одним или двумя параметрами
- С одним - возвращает значение по заданному ключу
- И удаляет его

УДАЛЕНИЕ КЛЮЧА

- В форме с двумя параметрами второй задает значение по умолчанию
- Если ключа нет, в первой форме бросается исключение
- Во второй - оно возвращается, исключения нет
- popitem - удаляет ранее всех созданный

УДАЛЕНИЕ КЛЮЧА

- Языковая конструкция -

```
del
```

- Может удалить переменную:

```
del v
```

УДАЛЕНИЕ КЛЮЧА

- Может удалить элемент списка: d

```
del data[5] #(дорого)
```

- Может удалить элемент кучи:

```
del d["key"]
```

ПРИМЕР

```
 1 data = {'vasya': 123, 'petya': 234}
 2 data['dima'] = 345
 3 kview = data.keys()
 4 print(kview)
 5 data['vasya'] = 345
 6 print(kview)
 7 del data['vasya']
 8 print(kview)
 9 data['vasya'] = 456
10 print(kview)
```

МАССИРОВАННОЕ ОБНОВЛЕНИЕ

- Общая идея
- Берем второй словарь
- И добавляем его содержимое в существующий
- Как будто прошли циклом по нему и сделали поэлементное добавление

ПРИМЕР

```
1 data = {'vasya': 123, 'petya': 234}
2 data_2 = {'petya': 345, 'misha': 456}
3
4 # изобретаем велосипед
5 for k, v: in data_2.items():
6     data[k] = v
7 print(data)
8 print(data_2)
```

МАССИРОВАННОЕ ОБНОВЛЕНИЕ

- Можно выполнить метод update:

```
data.update(data_2)
```

- Можно выполнить операцию

```
data | data_2
```

- Получим новый словарь, data останется как был

ПРИМЕР

```
 1 data = {'vasya': 123, 'petya': 234}
 2 data_2 = {'petya': 345, 'misha': 456}
 3 data.update(data_2)
 4 print(data)
 5 print(data_2)
 6
 7 data = {'vasya': 123, 'petya': 234}
 8 data_2 = {'petya': 345, 'misha': 456}
 9 print(data | data_2)
10 print(data)
11 print(data_2)
```

МАССИРОВАННОЕ ОБНОВЛЕНИЕ

- |= - близко по эффекту к update
- И предпочтительнее по стилю
- Но update позволяет использовать ключевые слова в параметрах
-
```
data.update(k=123, k2=234)
```

ПРОЧИЕ

- setdefault - близко по эффекту к get
- Только в случае отсутствия ключа устанавливает его в словаре
- В то значение, которое возвращает
- clear, copy, fromkeys, reversed

МНОЖЕСТВА

- Набор уникальных элементов
- Теоретически - неупорядоченный
- По факту - есть определенный порядок итерирования
- Но нет возможности обратиться по индексу

СОЗДАНИЕ

- Литерал - фигурные скобки (но есть нюанс)
- Функция-конструктор set (тоже есть нюанс)
- Множественное выражение: как словарное
- Только слева - один элемент, а не пара

ПРИМЕР

```
 1 data = {'vasya', 'petya'}
 2 print(data)
 3 data = {'vasya', 'petya', 'vasya'}
 4 print(data)
 5 data = {}
 6 print(data)
 7 print(type(data))  # !!!!!!
 8
 9 data = set(['vasya', 'petya'])
10 print(data)
11 #data = set('vasya', 'petya')
12 data = set('vasya')
13 print(data) # !!!!!
```

ПОЭЛЕМЕНТНОЕ ИЗМЕНЕНИЕ

- Метод add - добавляет элемент
- Метод remove - удаляет заданный элемент, бросает исключение, если его нет
- Метод discard - удаляет заданный элемент, если есть
- Метод pop - удаляет произвольный элемент, бросает исключение на пустом множестве

ВАЖНЫЙ МОМЕНТ

- Множество и словарь - родственные структуры
- Начиная с Python 3.6 реализована упорядоченность ключей словаря
- Начиная с 3.7 она прописана в спецификации
- Но в множестве - порядок не определен

ОПЕРАЦИИ НАД МНОЖЕСТВАМИ

- Как методы и как операции
- Изменяющие и не изменеяющие оригинал
- Объединение (неизменяющее): union или |
- Пересечение (неизменяющее): intersection или &

ОПЕРАЦИИ НАД МНОЖЕСТВАМИ

- Вычитание (неизменяющее): difference или -
- Симметрическая разность (неизменяющая): symmetric_difference или ^
- Изменяющие операции: |=, &=, -=, ^=
- Изменяющие методы: update, intersecton_update, и т.д.

ДРУГИЕ КОЛЛЕКЦИИ

- collections.Counter - счетчик (развитие словаря)
- collections.OrderedDict - вариации на тему порядка итерирования и удаления
- Помогает в структурах типа LRU
- collections.ChainDict - дерево словарей с делегированием родителю
- Помогает в fallback-стратегиях (например, при конфигурировании)

ДРУГИЕ КОЛЛЕКЦИИ

- collections.deque - совмещенный стек и очередь
- На циклическом буфере
- Примеры применения: скользящее окно, tail на потоке
- heapq.heapq - очередь с приоритетами
- Закидываем значения, умеем получать наименьшее

СТРУКТУРА ОПРЕДЕЛЕНИЯ

- Ключевое слово def
- Имя
- Параметры
- Тело

ПАРАМЕТРЫ

- Задаются в определении как перечень имен
- Привязка в точке вызова
- Можно по позиции, можно по имени
- Но можно ограничить способ задания параметра
  /ul>

ПРИМЕР

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

TRADEOFF

- Плюс именованных параметров - помогают не перепутать параметры
- Повышают наглядность кода
- Минус: становятся частью интерфейса
- Разнобой в традициях именования

ЧТО ПРЕДЛАГАЕТ PYTHON

- В момент определения функции можно выбрать стиль параметров
- Можно разный для разных параметров
- Вариант 1: чисто позиционные параметры
- Вариант 2: чисто именованные параметры
- Вариант 3: смешанные параметры

ЧТО ПРЕДЛАГАЕТ PYTHON

- В базовом варианте определения все параметры смешанные
- Можно в конце указать псевдопараметр /
- Тогда все параметры будут позиционными
- Или посередине
- Кто левее - позиционные, кто правее - смешанные

ПРИМЕР

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

ПРИМЕР

```
1 def is_near(a, b, /, epsilon):
2     return abs(a - b) < epsilon
3
4 print(is_near(1, 1.1, 0.01))
5 print(is_near(1, 1.0001, 0.01))
6 print(is_near(1, 1.0001, epsilon=0.01))
```

ЧТО ПРЕДЛАГАЕТ PYTHON

- Можно в начале указать псевдопараметр %
- Тогда все параметры будут именованными
- Или посередине
- Кто левее - смешанные, кто правее - именованные

ПРИМЕР

```
1 def find_car(*, year, mileage):
2     pass
3
4 #find_car(1995, 50000)
5 find_car(year=1995, mileage=50000)
6 find_car(mileage=50000, year=1995)
```

ПРИМЕР

```
1 def update_car(id, *, year, mileage):
2     pass
3
4 #update_car('qwertyuio', 1995, 50000)
5 find_car('qwertyuio', year=1995, mileage=50000)
6 find_car('qwertyuio', mileage=50000, year=1995)
7 find_car(id='qwertyuio', year=1995, mileage=50000)
8 find_car(id='qwertyuio', mileage=50000, year=1995)
```

В ОБЩЕМ ВИДЕ

- Можно использовать оба знака сразу
- Отсутствие знаков - это как будто начинается с % и заканчивается *
- Можно поставить рядом / и *
- Тогда не будет смешанных параметров