

PYTHON

Лекция 5

ПЛАН ЛЕКЦИИ

- Кортежи
- Словари
- Множества

КОРТЕЖИ

- Можно понимать как неизменяемые списки
- Можно понимать как короткий набор взаимосвязанных разнородных значений
- На практике - чаще во втором смысле

СОЗДАНИЕ КОРТЕЖА

- Как явная запись (литерал): в полной и сокращенной форме
- Вызов конструктора над существующей коллекцией
- Конструктор - tuple

ПРИМЕР

```
1 data = ('123', 'asdfgh', [])
2 print(data)
3 print(type(data), len(data))
4
5 data = '123', 'asdfgh', []
6 print(data)
7 print(type(data), len(data))
```

ПРИМЕР

```
1 data = ()
2 print(data)
3 print(type(data), len(data))
4
5 data = (123,)
6 print(data)
7 print(type(data), len(data))
```

РАЗБЕРЕМ

- В полной форме - как список, только с круглыми скобками
- В сокращенной - без скобок
- Сокращенная разрешена в тех местах, где нет неоднозначности
- Пример неоднозначности: хотим передать в функцию один параметр-кортеж
- Без скобок будет понято как много параметров

РАЗБЕРЕМ

- Пустой кортеж можно указать только со скобками
- Кортеж из одного элемента даже в скобках вызывает неоднозначность
- Понимается просто как элемент в скобках
- Для распознавания его как кортежа нужна "висящая" скобка
- Там, где можно использовать кортеж без скобок - можно и одноэлементный без скобок

ГРУППОВОЕ ПРИСВАИВАНИЕ

- Вспомним запись вроде $a, b = b, a$
- Или $a, b = 0, 1$
- По сути, справа от присваивания - кортеж

ИДИОМАТИЧНОЕ ИЗВЛЕЧЕНИЕ ПОЛЕЙ

- К нам приехала переменная, в ней кортеж
- Нам нужны его поля
- Например, имя, номер паспорта и дата рождения
- Идиоматично написать так: `name, passport, birthdate = person`
- Не извлекать поэлементно через явный индекс

ИДИОМАТИЧНОЕ ИЗВЛЕЧЕНИЕ ПОЛЕЙ

- Если поле заведомо ненужно - можно вместо имени использовать подчеркивание
- Если одно из полей - тоже кортеж, можно использовать вложенность
- Например, пусть дата рождения - тоже кортеж
- И нам нужно имя и год рождения
- `name, _, (_, _, year) = person`

ИДИОМАТИЧНОЕ ИЗВЛЕЧЕНИЕ ПОЛЕЙ

- А если полей 15 штук ?
- И нужно второе и четвертое ?
- Как-то оно странно будет смотреться

ИДИОМАТИЧНОЕ ИЗВЛЕЧЕНИЕ ПОЛЕЙ

- Если полей больше, чем 4-5, надо использовать именованные кортежи
- Или data-классы
- И что-то серьезное в кортежах тоже лучше не хранить
- Легко запутаться, где какое поле
- Они скорее для того, чтобы что-то вернуть из функции, передать и т.п.

ПРИМЕР

```
1 data = tuple('123')
2 print(data)
3 print(type(data), len(data))
4
5 data = tuple([123, 321, 0, None])
6 print(data)
7 print(type(data), len(data))
8
9 data = tuple(data)
10 print(data)
11 print(type(data), len(data))
```

ОПЕРАЦИИ, МЕТОДЫ

- Индексы, вырезки - как у строк/списков
- Методы - подмножество методов списка
- Нет тех, что меняют состояние
- И сору - потому что нет смысла
- В остатке - count и index
- Стандартное итерирование

СЛОВАРИ

- Чисто формально "как словарь, только индекс - строка"
- Но не обязательно строка, а любой неизменяемый тип данных
- Нет понятия "следующий индекс"
- Принцип хранения данных - хеш-таблица

СОЗДАНИЕ СЛОВАРЯ

- Можно создать пустой словарь как литерал: {}
- Или как вызов функции конструктора: dict()
- Можно внутри фигурных скобок указать пары ключ-значение
- Пары отделяются запятыми, ключ от значения - точкой с запятой

СОЗДАНИЕ СЛОВАРЯ

- А еще можно передать в dict любую коллекцию пар
- Обычно это список 2-элементных кортежей
- Но не обязательно

ПРИМЕР

```
1 data = {'vasya': 123, 'petya': 234}
2 print(data)
3 print(type(data), len(data))
4 data = {'vasya': 123, 'vasya': 234}
5 print(data)
6 print(type(data), len(data))
7 data = {}
8 print(data)
9 print(type(data), len(data))
10 data = dict()
11 print(data)
12 print(type(data), len(data))
```

ПРИМЕР

```
1 data = {('vasya', 'petya'): 123, ('petya', 'vasya'): 234}
2 print(data)
3 print(type(data), len(data))
4
5 #data = {['vasya', 'petya']: 123, ['petya', 'vasya']: 234}
6 # так нельзя
```

СЛОВАРНОЕ ВЫРАЖЕНИЕ

- По аналогии со списочным выражением есть словарное
- Общая структура - как у списочного
- Только в фигурных скобках
- А справа от `for` пишем пару, разделенную двоеточием

ПРИМЕР

```
1 data = {u.id: calculate_rating(u)
2         for u in users if user.login_count}
3
4 # Лучше, чем так:
5 data = {}
6 for u in users:
7     if user.login_count:
8         data[u.id] = calculate_rating(u)
```

А ЧТО С КОРТЕЖАМИ ?

- Есть списки и списочные выражения
- Есть словари и словарные выражения
- А как насчет кортежей ?
- И кортежных выражений с круглыми скобками

А ЧТО С КОРТЕЖАМИ ?

- Конструкция с круглыми скобками есть
- И мы до нее доберемся
- Но она не про кортежи
- Но можно написать так:

```
tuple(v for v in data)
```


ДОБАВЛЕНИЕ/ИЗМЕНЕНИЕ

- Классическое индексное присваивание
- Порядок создания ключей определяет порядок итерирования
- Обновление существующего ключа - не меняет порядка

ИЗВЛЕЧЕНИЕ

- Два способа: индекс в квадратных скобках и метод `get`
- Разница - в реакции на отсутствующие ключи
- Индексная форма бросает исключение
- Метод `get` возвращает `None`
- Метод `get` со вторым параметром возвращает второй параметр

ПРИМЕР

```
1 data = {}
2 #print(data['vasya'])
3 # исключение
4
5 print(data.get('vasya')) # None
6 print(data.get('vasya', 'nothing')) # nothing
7
8 data['vasya'] = 123
9 print(data['vasya']) # 123
10 print(data.get('vasya')) # 123
11 print(data.get('vasya', 'nothing')) # 123
```

ПРИМЕР

```
1 data = {'vasya': None}
2 print(data.get('vasya')) # None
3 print(data.get('petya')) # None
4
5 print(data.get('vasya', 'nothing')) # None
6 print(data.get('petya', 'nothing')) # nothing
```

ПРОВЕРИТЬ НАЛИЧИЕ

- Простой и надежный способ - операция in
- И парная ей - not in
- Вернет признак наличия/отсутствия ключа
- Неважно - что там лежит

ИТЕРИРОВАНИЕ

- Итерирование словарю - это итерирование по ключам
- Можно итерироваться по значениям
- Или по парам ключ-значение
- Структурно менять словарь во время итерирования не стоит

ПРИМЕР

```
1 data = {('vasya', 'petya'): 123, ('petya', 'vasya'): 234}
2
3 for k in data:
4     print(k)
5
6 for k in data: # не идиоматично
7     print(data[k])
8
9 for k in data: # не идиоматично
10    print(k, data[k])
```

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 изменяющие 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`, `intersection_update`, и т.д.

