eebb

ЯЗЫК PYTHON

Лекция 9. Кортежи, множества

Кортежи

- Как списки, только неизменяемые
- Синтаксически отличаются скобками: круглые вместо квадратных
- Можно создавать, извлекать элементы и вырезки
- Можно склеивать, создавая новые

Синтаксические нюансы

- Круглые скобки еще используются для выражений
- А еще для вызова функций
- И еще для ленивых генераторов (пока не спрашивайте, что это)
- Что создает некоторую путаницу

Синтаксические нюансы

- Если список пустой, то обозначаем []
- А если кортеж пустой - то ()
- Для пустого кортежа скобки нужны всегда
- Нет другого способа отличить пустой кортеж от случайно пропущенного значения

Синтаксические нюансы

- Если в списке 1 элемент, то обозначаем [123]
- А если в кортеже 1 элемент - то без скобок тоже нельзя
- Нет способа отличить 1-элементный кортеж от простого значения
- Но и скобок - недостаточно

Синтаксические нюансы

- Потому что одна из законных форм выражения - это подвыражение в скобках
- Если бы мы ничего не знали про кортежи, мы бы (5) + (4) распознали как 5 + 4
- Только немного странно записанное
- Чтобы различать - добавляем запятую после значения
- (5,) + (4,) - конкатенация кортежей

Без нюансов

- Начиная с размера 2, нюансы заканчиваются
- Кортеж всегда можно представить перечислением значений
- Через запятую в круглых скобках
- Где контекст позволяет - и без скобок

Пример: кортежи

```
1 v = 1, 2, 3  # кортеж длиной 3
2 v = (1, 2) # кортеж длиной 2
3 v = (1,)   # кортеж длиной 1
4 v = 1,     # кортеж длиной 1
5 v = (1)    # просто число 1
6 v = 1      # просто число 1
7 v = ()     # кортеж длиной 0
8 v =        # синтаксическая ошибка
```

Еще про запятую в конце

- Если длина кортежа - больше, чем 1
- Все равно можно в конце написать запятую
- И в списке - тоже
- Это удобно, чтобы делать copy-paste в длинных структурах

Пример: висящая запятая

```
1 data = (
2     123,
3     234,
4     345,
5 )
6 print(data)
7 print(len(data))
```

Где нужны скобки

- Там, где запятые имеют другой смысл
- Например, при вызове функции
- Или при задании списка литералом
- [1, 2] - это список двух чисел
- [(1, 2)] - это список одного кортежа

Применения кортежей: теория

- Как список, который не нужно менять
- Хочется обезопасить себя от изменений
- Или хочется делать много больших вырезок из больших списков
- Или перечень элементов хочется хранить в множестве как единое целое

Применения кортежей: практика

- Реально они редко используются как неизменяемые списки
- Чаще критерий для выбора списка или кортежа другой
- Когда много однородных элементов - используются списки
- Когда мало разнородных - кортежи

Применения кортежей: примеры

- Если храним id пользователя и имя пользователя вместе, то используем пару
- Т.е. кортеж из двух элементов
- А если храним список пользователей сервиса
- То скорее будем хранить в списке
- Даже если этот список не меняется
- Это практика, но иногда полезно ее подвергнуть сомнению

Групповое присваивание

- Пусть справа от присваивания - кортеж
- Слева может быть одна переменная
- Или ровно столько, сколько элементов в кортеже
- В первом случае весь кортеж присваивается переменной
- Во втором - каждое значение присваивается соответствующей переменной

Особый случай

- Если справа кортеж из одного элемента
- Слева можно указать просто одну переменную
- И присвоить кортеж
- А можно указать переменную с запятой
- И тогда присвоится значение без кортежа

Примеры присваиваний

```
 1 a, b = 1, 2
 2 print("a", a) # a 1
 3 print("b", b) # b 2
 4
 5 a, b = b, a
 6 print("a", a) # a 2
 7 print("b", b) # b 1
 8
 9 c = 1, 2
10 print("c", c) # c (1, 2)
```

Примеры присваиваний (продолжение)

```
1 a, b = c[::-1]
2 print("a", a) # a 2
3 print("b", b) # b 1
4
5 a, b = c, (a, b)
6 print("a", a) # a (1, 2)
7 print("b", b) # b (2, 1)
```

Пересечение переменных справа и слева

- Простое правило
- Сначала считаем все, что справа
- Потом присваиваем поэлементно

Примеры с одноэлеметным кортежем

```
 1 a = 1,
 2 print("a", a) # a (1,)
 3
 4 a, = 1,
 5 print("a", a) # a 1
 6
 7 data = (5, 10, 20)
 8 a = data[:1]
 9 print("a", a) # a (5,)
10
11 a, = data[:1]
12 print("a", a) # a 5
```

Примеры ошибок

```
1 a, b = 1, 2, 3     # слишком много справа
2 a, b, c = 1, 2     # слишком мало справа
3 a = 'hello'
4 a, b = a[:1]       # слишком мало справа
5 a, = ()            # слишком мало справа
6 a, = 4             # скалярное значение справа
7 x, a = 1, x        # обращение к неприсвоенной переменной
```

Специальная переменная _

- Иногда есть кортеж
- И нам интересны некоторые его значения
- Но не все
- Можно использовать _ качестве заполнителя

Пример с заполнителями

```
1 data = [(1, 2), (2, 3), (12, 9)]
2
3 for _, second in data:
4     print(second)
```

Не только про кортежи

- a, b = 1, 10
- Справа - кортеж (1, 10)
- a, b = data
- data - может быть и не кортежем
- Может быть списком
- Главное - чтобы два элемента

Не только про кортежи

- Но скорее это будет кортеж
- Потому что списки часто меняются в размере
- a, b = [1, 10]
- Так тоже можно, но не идиоматично

Идиомы

- a, b = input().split()
- Если гарантированы ровно два элемента
- a, b = input().split(2)
- Если гарантированы не менее двух элементов

Пример на идиоматичность

```
 1 point_3d = (1, 10, 20)
 2
 3 # Если уверены, что в data - 3 элемента, то лучше
 4
 5 x, y, z = point_3d
 6
 7 # чем
 8
 9 x = point_3d[0]
10 y = point_3d[1]
11 z = point_3d[2]
```

Вложенные групповые присваивания

- Пусть есть кортеж
- Имя, id, и дата рождения
- Дата рождения - тоже кортеж
- День, месяц, год
- Можно слева использовать вложенность

Пример на вложенные групповые присваивания

```
 1 user = ('vasya', 'id-1234567', (10, 5, 2001))
 2
 3 name, id, (day, month, year) = user
 4 print(name, id, day, month)
 5
 6 name, _, (day, _, year) = user
 7 print(name, day, year)
 8
 9 name, id, birthdate = user
10 print(name, id, birthdate)
```

Кортеж как неизменяемая последовательность

- Список - это изменяемая последовательность элементов
- Кортеж - это неизменяемая последовательность элементов
- Строка - это неизменяемая последовательность символов
- Все, что есть у списков и строк как у последовательностей - есть и у кортежей
- То, чего нет у строк в силу их неизменяемости - нет и у кортежей

Кортеж как неизменяемая последовательность

- Индексы, отрезки в режиме чтения - есть
- Изменения элементов - нет
- Изменения отрезков - нет
- Сложение, умножение на число - есть
- Перебор в цикле - есть

Множество

- Для данных о пользователе удобно использовать кортеж
- Для данных о многих пользователях - список (кортежей)
- А если хотим знать, какие имена присутствуют у пользователей ?
- Каких годов рождения они бывают ?
- В целом, без привязки к конкретным личностям
- Или - какие слова встречаются в тексте

Множество

- Есть отдельный вид коллекции - множество
- В множества элементы можно добавлять
- Или удалять
- Повторное добавление того же элемента не меняет множества
- Одно удаление дважды добавленного элемента удаляет его полностью

Синтаксис

- Непустое множество можно задать перечислением элементов
- В фигурных скобках
- Пустые фигурные скобки задают пустой словарь (о них не сегодня)
- Пустое множество можно получить вызовом set()
- Пустой список - list(), пустой кортеж - tuple(), пустую строку - str()

Примеры множеств

```
 1 empty = set()
 2
 3 single_number = {123}
 4 single_string = {'123'}
 5 print(single_number, single_string)
 6
 7 two_numbers = {123, 234}
 8 two_strings = {'123', '234'}
 9 print(two_numbers, two_strings)
10
11 two_elems = {123, '123'}
12 two_elems_again = {123, '123', 123}
13 print(two_elems, two_elems_again)
```

Изменяемость

- Множества изменяемы
- Есть методы add и remove
- Они добавляют и удаляют элементы
- Изменяют множество, над которым вызваны

Пример на добавление

```
1 names = set()
2
3 names.add('vasya')
4 print(names)
5 names.add('petya')
6 print(names)
7 names.add('vasya')
8 print(names)
```

Пример на удаление

```
1 names = set()
2
3 names.add('vasya')
4 names.add('petya')
5 print(names)
6 names.remove('vasya')
7 print(names)
8 names.remove('kolya') # будет исключение
9 print(names)
```

Операции теории множеств

- Реализованы в двух вариантах
- В неизменяемом (порождаем новое множество)
- В изменяемом (изменяем одно из множеств)
- Неизменяемый вариант представлен как операции

Операции теории множеств

- Объединение: |
- Пересечение: &
- Разность: -
- Симметрическая разность: ^

Примеры операций

```
1 names_1 = {'vasya', 'petya', 'dima'}
2 names_2 = {'petya', 'kolya', 'sasha', 'dima'}
3
4
5 print(names_1 | names_2)
6 print(names_2 & names_2)
7 print(names_1 - names_2)
8 print(names_2 - names_1)
9 print(names_1 ^ names_2)
```

Добавление/удаление в неизменяемом стиле

- Формально нет
- Но можно придумать замену
- Добавление: values | {new_value}
- Удаление: values - {to_delete}

Операции в изменяемом стиле

- Есть методы
- С очень громоздкими имена
- intersection_update
- union_update
- symmetric_difference_update
- difference_update

Примеры операций

```
1 names_1 = {'vasya', 'petya', 'dima'}
2 names_2 = {'petya', 'kolya', 'sasha', 'dima'}
3 names_1.update(names_2)
4 print(names_1)
5
6 names_1 = {'vasya', 'petya', 'dima'}
7 names_2 = {'petya', 'kolya', 'sasha', 'dima'}
8 names_2.intersection_update(names_1)
9 print(names_2)
```

Примеры операций

```
1 names_1 = {'vasya', 'petya', 'dima'}
2 names_2 = {'petya', 'kolya', 'sasha', 'dima'}
3 names_1.difference_update(names_2)
4 print(names_1)
5
6 names_1 = {'vasya', 'petya', 'dima'}
7 names_2 = {'petya', 'kolya', 'sasha', 'dima'}
8 names_2.symmetric_difference_update(names_1)
9 print(names_2)
```

Преобразования типов коллекций

- У каждой есть функция-конструктор
- set, str, tuple, list
- Можем передать коллекцию в конструктор другого типа
- Например set(['abc', 'bca', 'abc'])
- Или list({'abc', 'bca'})

Преобразования типов коллекций

- Для того, что передаем в set, перебираем элементы
- И добавляем в множество
- Для строки - добавляем символы
- Если set передаем в list/tuple
- Создаем список или кортеж из элементов множества
- В порядке их добавления в множество

Примеры преобразований

```
1 names = set(['vasya', 'petya', 'dima'])
2 print(names)
3
4 names = set(('petya', 'kolya', 'sasha', 'dima'))
5 print(names)
6
7 letters = set(''.join(['vasya', 'petya', 'dima']))
8 print(letters)
```

Примеры преобразований

```
1 names = set(['vasya', 'petya', 'dima'])
2 for v in ('petya', 'kolya', 'sasha', 'dima'):
3     names.add(v)
4
5 print(list(names))
6 print(tuple(names))
```