Питон
Курс лекций
Лекция пятая

© Зайцев Вадим Валерьевич, 2002–2010,
spider.vz@gmail.com

Разобравшись с кавычками, мы можем перейти к скобкам. Их существует четыре вида: круглые, квадратные, фигурные и угловые. Три из них служат для определения трёх важнейших сложных типов данных питона.

3.9 Композитные типы данных

Существует два сложных, или композитных, типов данных в питоне: последовательности и объекты. Сегодня мы разберёмся с тремя разновидностями последовательностей.

Определение. Последовательность есть нечто, заключённое в скобки.

1. Кортеж *есть неоднородный неизменяемый массив. Задаётся круглыми скобками или же их отсутствием.* Ну, неоднородный — это понятно, значит, может содержать разнотиповые данные, например:

   ```
   A=(2,3.14,"aaa")
   ```

   ```
   B=((((((1),0),0),0),0),0)
   ```

   Неизменяемый — это сложнее. Это значит, что структура кортежа не может быть изменена после того, как он был создан. (Как будет выяснено далее, кое-что можно всё же сделать в обход ограничений). В питоне только строки и кортежи являются неизменяемыми типами данных. Так, нельзя заменить одну букву в строке, оставив саму строку той же, но можно создать новую строку с одной изменённой буквой.

   Для доступа к элементам кортежа используются квадратные скобки с указанием номера нужного элемента:

   ```
   C=A[1]
   ```

   В этом случае в `C` будет занесена не двойка, а 3.14, потому что *нумерация элементов всегда идёт с нуля.* Также можно из кортежа взять

часть с несколькими элементами, называемую сечением. Сечения бывают трёх видов: начальные, центральные и конечные. Рассмотрим различия между ними на примерах:

`D=(1,2,3,4,5,6,7,8,9)` даёт (1,2,3,4,5,6,7,8,9)

`E=D[3:8]` даёт (4,5,6,7,8)

`F=D[:4]` даёт (1,2,3,4)

`G=D[7:]` даёт (8,9)

В первой строчке мы занесли в `C` какой-то произвольный кортеж, удобный для демонстрации различных сечений. Во второй строчке берётся центральное сечение — с третьего элемента включительно по восьмой невключительно. Следует отметить, что это обычный для питона метод обхождения с границами чего бы то ни было — нижняя граница всегда входит в диапазон, а верхняя — нет. Это не обусловлено никакими теоретическими выкладками, а только практическим удобством использования. Ну и, конечно, ни на минуту нельзя забывать, что нумерация элементов идёт с нуля! В третьей строчке мы опустили первое число, и оно по умолчанию приняло значение 0 — номер первого элемента, что дало нам начальное сечение. Ясно, что конечное сечение получается при опускании последнего индекса, принимающего номер на один больший номера последнего элемента (то есть так, чтобы последний элемент вошёл в сечение, а не остался непонятно где).

У некоторых логично мыслящих может возникнуть вопрос: а что, если опустить оба числа? Правильный ответ таков: результатом будет полное сечение или копия исходного кортежа. Такой ответ ожидаем, но не вносит ясности, появления которой мы так жаждали при формулировке вопроса, и даже наоборот, он запутывает ситуацию, порождая новый вопрос: в чём разница между `H=D` и `H=D[:]`? Ответ: **в семантике!**

Дело в том, что в питоне для сложных типов данных (то есть не строк и не чисел) оператор присваивания работает совсем по-другому. Вместо пересылки содержимого одних ячеек памяти в другие происходит дополнительное именование *тех же самых* ячеек. Таким образом, разные для нас имена трактуются как один и тот же набор ячеек питоном. Это называется семантика указателей.

Количество имён объекта[1] называется его мощностью. Для уменьшения мощности используется оператор `del`. Когда мощность объекта опускается до нуля, объект потерян, мы больше не имеем к нему доступа, и в ближайшее время он будет уничтожен интерпретатором питона. Все строки и числа имеют мощность 1 и уничтожаются сразу по вызову оператора `del` или при получении именем нового значения. Запись `H=D[:]` олицетворяет семантику копирования. Создаётся новый объект, полностью копирующий структуру и содержимое старого, и

[1] Объектом мы пока что называем множество ячеек памяти

мы получаем два одинаковых (точнее, разных, но равных) объекта мощности 1 каждый.

Для преобразования строки или последовательности в кортеж используется функция `tuple()`:

```
>>> tuple('123')
('1','2','3')
```

Если аргумент этой функции — кортеж, она вернёт именно его (а не его копию).

2. Список *есть изменяемый неоднородный массив. Задаётся квадратными скобками.*

   Список — это более обычный для опытных (испорченных другими языками) программистов, встречающийся во многих языках программирования высокого уровня. Сечения берутся подобным же образом:

   `K=range(1,10)` даёт [1,2,3,4,5,6,7,8,9]

   `L=K[2:]` даёт [3,4,5,6,7,8,9]

   В первой строчке стандартной функцией питона мы создали список последовательных целых чисел от 1 до 10 (как обычно, первое число вошло в результат, а второе — нет). Эта функция чрезвычайно полезна и, как сказал бы на нашем месте философ, если бы её не было, её стоило бы придумать. Полную её мощь мы сможем вкусить на следующей лекции, когда доберёмся до операторов циклов. Пока же вернёмся к последовательностям.

   Во второй строчке берётся конечное сечение — с третьего элемента по последний включительно. Прочие типы сечений берутся аналогично.

   Кроме того, мы можем изменять значения элементов списка:

   `K[5]=5` даёт в K [3,4,5,6,7,5,9]

   Здесь следует быть осторожным и различать семантику копирования и семантику указателей. Например, если мы напишем:

   ```
   M=[3,4,5]
   N=M
   N[0]=333
   ```

   То какое значение окажется в `M`? Правильно, `[333,4,5]`, потому что какое бы имя мы не использовали: `M` или `N`, обращение идёт к одним и тем же ячейкам оперативной памяти.

   Воспользуемся уже известным методом для определения операций над списком — функцией `dir`:

   ```
   >>> dir([])
   ['append', 'count', 'extend', 'index', 'insert', 'pop',
   'remove', 'reverse', 'sort']
   ```

Эти функции служат соответственно для добавления элемента в конец готового списка; для подсчёта количества элементов списка, равных данному; для расширения этого списка другой последовательностью (в т.ч. и кортежем) добавлением всех его элементов в конец этого; для действия, обратного взятию элемента — определения индекса элемента по его значению; `N.insert(2,3)` позволяет вставить новый элемент не в конец списка, а на определённое место, номер места идёт первым параметром, значение — вторым; `pop` позволяет *вытолкнуть* последний элемент из списка (при этом сам список становится короче); `remove` удаляет элемент, значение которого ему дано; `reverse` обращает список (последний элемент теперь идёт первым); `sort` сортирует список по возрастанию. Для сортировки по убыванию, конечно, можно использовать комбинацию `sort` и `reverse`.

Упражнение. Запустите функцию `dir` для кортежа и объясните результат.

Для преобразования строки или последовательности в список используется функция `list()`:

```
>>> list('123')
['1','2','3']
>>> list((1,2,3))
[1,2,3]
```

Если аргумент этой функции — список, то она создаст копию и вернёт именно её (а не его исходный список). Таким образом, если `P` — список, то `P=list(Q)` эквивалентно `P=Q[:]`.

Почему для кортежа копия не создавалась, а для списка создаётся? Да просто никто не будет изменять кортеж, и нет смысла хранить два одинаковых неизменяемых объекта.

Вдумчивому слушателю (читателю) не даст покоя очередной вопрос: имеют ли кортежи право на существование, раз всё их отличие от списков заключается в неудобстве, связанном с невозможностью изменения структуры? Разница между кортежем и списком философская, её можно уподобить разнице между джазом и блюзом. И то, и другое — музыка, которую могут исполнять одни и те же музыканты на тех же инструментах, но разницу можно уловить невооружённым ухом. Суть этих стилей принципиально разная: джаз — это импровизация, полёт фантазии, экспромт, а блюз — это крик души испытываемого судьбой человека. Список — это прежде всего нумерованая последовательность, кортеж — прежде всего упорядоченная. Координаты точки в пространстве — это кортеж; перечень фамилий студентов — список. Цветовые составляющие пикселя — кортеж; строчки, введённые из файла — список. Конечно, удобства и полноты ради и список упорядочен, и кортеж пронумерован, но это уже вторично. Безусловно,

никто (и даже сам Гвидо ван Россум) не в силах вам помешать использовать списки вместо кортежей везде, где они только встречаются — начинающие программисты, замученные паскалем, бейсиком и явой, так и делают, но это (кроме замедления работы программы) есть ни что иное, как преступление против философии — самое тяжкое из всех преступлений, а если это и что-то другое, то демонстрация собственного невежества, некомпетентности и неумения грамотно пользоваться предоставляемыми средствами. Тем не менее, даже дав зарок никогда не пользоваться кортежами, вы рискуете вскоре его нарушить, даже не подозревая об этом. Например, в следующем случае:

```
R,S=2,3
```

В этом случае на лету создаётся кортеж, прозрачный для программиста, и тут же уничтожается за ненадобностью. Нет необходимости ни нумеровать элементы (кроме как для того, чтобы знать, в какой последовательности они идут), ни реализовывать возможность последующего изменения элементов, их добавления, сортировки и т.д. — легко, просто, быстро, доступно. Аналогично используется так называемая декомпозиция кортежа:

```
T=1,2,3
```

```
U,V,W=T
```

Теперь понятно, как реализуется предыдущий пример: кортеж сначала создаётся, а потом декомпозиционируется.

А о каких возможностях мы говорили, заявляя, что как-то можно обойти ограничения, наложеные изобретателем кортежей? Есть одна лазейка в его **определении**, позволяющая кое-что сделать. Сказано — нельзя менять структуру. Изменение самих элементов кортежа может существенно изменить структуру, потому и запрещено. Но представьте, что один из элементов кортежа — список. Можно ли менять его элементы? Да, конечно!

```
X=[2,3,4],5,6
```

`X[1]=4` — нельзя

`X[0][1]=33` — можно

3. Словарь *есть ассоциативный изменяемый неоднородный массив. Задаётся парами ключ: значение, перечисленными в фигурных скобках.*

   Словарь — это квинтэссенция программистской мысли, направленной на изучение последовательностей, это массив, в котором элементы пронумерованы не подряд идущими целыми положительными числами, а чем угодно: строками, вещественными числами, кортежами. Конечно, это не может быть сделано списками (которые меняют свою структуру) и словарями. Индекс элемента словаря называется ключом.

Сечений словарь не поддерживает, а для создания копии нужно написать в явном виде:

```
Y={"one":1,"two":2}
```

```
Z=A.copy()
```

Полное очищение словаря производится функцией `clear()`, добавление новой пары — простым присваиванием:

```
Y["three"]=3
```

При попытке узнать значение по несуществующему ключу выдаётся ошибка и работа программы останавливается, поэтому нужно почаще пользоваться функцией `has_key()`, определяющей, есть ли такой ключ в данном словаре. Кроме получения списка ключей (`keys()`) и списка значений (`values()`) весьма полезно **представление словаря как списка кортежей** функцией `items()`. Понятно, что здесь философия соблюдена — легко что-то добавить в полученный *псевдокортеж* или отсортировать его, а каждой паре это ни к чему — всё, что нам нужно знать, это где ключ и где значение, ему соответствующее. Это с лихвой обеспечивается кортежем.

Прочие возможности и приёмы работы со словарём можно узнать у функции `dir({})`.

Ясно, что уже данное нами определение типа при нынешнем уровне знаний не выдерживает никакой критики, и нужно давать его заново и подругому:

Определение. Тип есть совокупность множества значений и методов для работы с ними.

Упражнение. Множество допустимых значений типа — это список или кортеж? А сам тип?

Существует большое разнообразие типов, необходимых в самых разных областях применения программирования. Среди наиболее интересных можно вспомнить множества — когда важно присутствие элемента, но не важен его порядок, и один элемент может присутствовать только в единственном экземпляре. Для графов существует много разных машинных представлений: матрицы инцидентности и смежности, объединение множества дуг и множества вершин и т.п. Деками называют массивы, в которых доступ может производиться только к крайним элементам, по одному с каждой стороны (такая же, но одностороняя структура называется стеком). В некоторых задачах удобно пользоваться кольцами — замкнутыми массивами, в которых доступ осуществляется только к одному элементу кольца и для перехода к другому кольцо нужно *прокрутить*. Всё это — сильно специализированные вещи, не входящие в стандартную поставку питона, но их можно реализовать с помощью объектной модели, о чём мы и узнаем через несколько лекций.