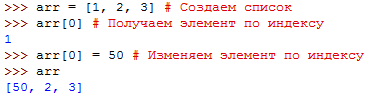
**Списки и кортежи**

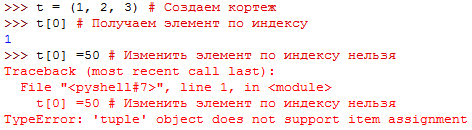
Списки и кортежи - это нумерованные наборы объектов. Каждый элемент набора содержит лишь ссылку на объект. По этой причине сnиски и кортежи могут содержать объекты произвольного типа данных и иметь неограниченную степень вложенности. Позиция элемента в наборе задается индексом. Обратите внимание на то, что нумерация элементов начинается с 0, а не с 1.

Списки и кортежи являются уnорядоченными nоследовательностями элементов. Как и все последовательности, они nоддерживают обращение к элементу по индексу, nолучение среза, конкатенацию (оnератор **+**), повторение (оnератор **\***), проверку на вхождение (оnератор **in**).

Списки относятся к **изменяемым** тиnам данных. Это означает,что можно не только nолучить элемент по индексу, но и изменить его:



Кортежи относятся к **неизменяемым** тиnам данных. Иными словами, можно получить элемент по индексу, но изменить его нельзя:

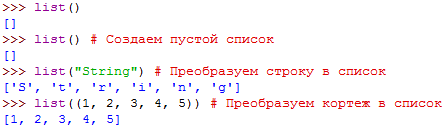


**8.1. Создание списка**

Создать сnисок можно следующими способами:

* с помощью функции **list([<Последовательность>])**. Функция позволяет преобразовать любую последовательность в список. Если параметр не указан, то создается пустой список.

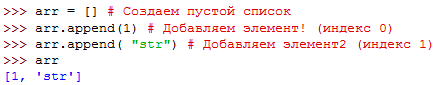
Пример:



* указав все элементы списка внутри квадратных скобок:



* заполнив список поэлементно с помощью метода append():



В некоторых языках программирования (наnример, в РНР) можно добавить элемент, указав пустые квадратные скобки или индекс больше последнего индекса. В языке **Python** все эти сnособы nриведут к ошибке.

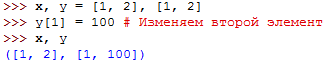
При создании списка в переменной сохраняется ссылка на объект, а не сам объект. Это обязательно следует учитывать при групповом nрисваивании. Групповое nрисваиванне можно использовать для чисел и строк, но для списков этого делать нельзя. Рассмотрим пример:



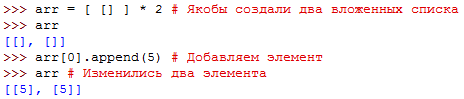
В этом nримере мы создали список из двух элементов и nрисвоили значение переменным х и у. Теперь попробуем изменить значение в переменной у:



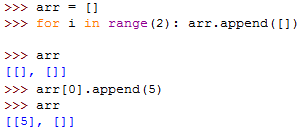
Как видно из примера, изменение значения в переменной у привело также к изменению значения в переменной х. Таким образом, обе переменные ссылаются на один и тот же объект, а не на два разных объекта. Чтобы получить два объекта, необходимо производить раздельное присваивание:



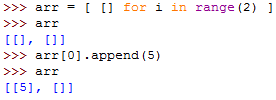
Точно такая же ситуация возникает при использовании оператора повторения \*. Например, в следующей инструкции производится поnытка создания двух вложенных списков с помощью оператора \*:



Создавать вложенные списки следует с nомощью метода **append()** внутри цикла:

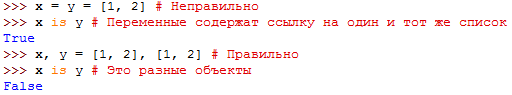


Можно также воспользоваться генераторами списков:

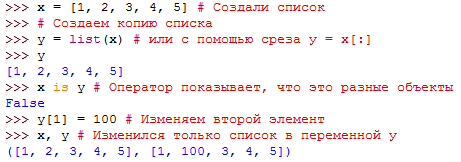


Проверить, ссылаются ли две переменньrе на один и тот же объект, nозволяет оnератор is.

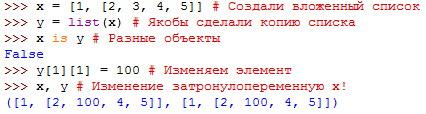
Если переменные ссылаются на один и тот же объект, то оnератор **is** возвращает значение **True**:



Как вы уже знаете, операция присваивания сохраняет лищь ссылку на объект, а не сам объект. Что же делать, если необходимо создать копию сnиска? Первый способ заключается в применении операции извлечения среза, а второй способ - в исnользовании функции **list()** (листинг 8.1 ).

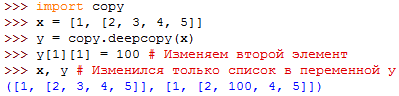


На первый взгляд может показаться, что мы получили копию. Оператор **is** показывает, что это разные объекты, а изменение элемента затронуло лишь значение переменной **у**. В данном случае вроде все нормально. Но проблема заключается в том, что списки в языке **Python** могут иметь неограниченную степень вложенности. Рассмотрим это на примере:

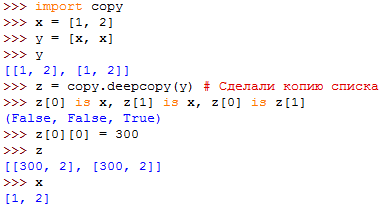


В этом примере мы создали список, в котором второй элемент является вложенным списком. Далее с помощью функции **list()** попытались создать копию списка. Как и в предыдущем примере, оператор **is** показывает, что это разные объекты, но посмотрите на результат. Изменение переменной у затронуло и значение переменной х. Таким образом, функция **list()** и операция извлечения среза создают лишь поверхностную копию списка.

Чтобы получить полную копию списка, следует воспользоваться функцией **deepcopy()** из модуля **сору** (листинг 8.2).



Функция **deepcopy()** создает копию каждого объекта, при этом сохраняя внутреннюю структуру списка. Иными словами, если в списке существуют два элемента, ссылающиеся на один объект, то будет создана копия объекта, и элементы будут ссылаться на этот новый объект, а не на разные объекты. Пример:



Значение изменилось сразу в двух элементах! Начальный список не изменился.

**8.2. Операции над списками**

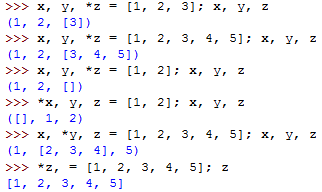
Обращение к элементам списка осуществляется с помощью квадратных скобок, в которых указывается индекс элемента. Нумерация элементов списка начинается с нуля. Выведем все элементы списка:



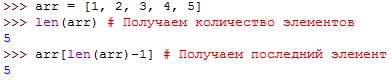
С помощью позиционного присваивания можно присвоить значения элементов списка каким-либо переменным. Количество элементов справа и слева от оператора = должно совпадать, иначе будет выведено сообщение об ошибке:



В **Python 3** при позиционном присваивании перед одной из переменных слева от оператора = можно указать звездочку. В этой переменной будет сохраняться список, состоящий из "лишних" элементов. Если таких элементов нет, то список будет пустым:



Так как нумерация элементов списка начинается с 0, индекс последнего элемента будет на единицу меньше количества элементов. Получить количество элементов списка позволяет функция **len()**:



Если элемент, соответствующий указанному индексу, отсутствует в списке, то возбуждается исключение **IndexError**.

В качестве индекса можно указать отрицательное значение. В этом случае смещение будет отсчитываться от конца списка, а точнее сказать, значение вычитается из общего количества элементов списка, чтобы получить положительный индекс:



Так как списки относятся к изменяемым типам данных, то мы можем изменить элемент по индексу.

Кроме того, списки поддерживают операцию извлечения среза, которая возвращает указанный фрагмент списка.

Формат операции:

**[<Начало>:<Конец>:<Шаг>]**

Все параметры являются необязательными. Если параметр <**Начало**> не указан, то используется значение 0. Если параметр <**Конец**> не указан, то возвращается фрагмент до конца списка. Следует также заметить, что элемент с индексом, указанном в этом параметре, не входит в возвращаемый фрагмент. Если параметр <**Шаг**> не указан, то используется значение **1**.

В качестве значения параметров можно указать отрицательные значения.

Выведем элементы в обратном порядке:



Выведем список без первого элемента:



Выведем список без последнего элемента



Получим первые два элемента списка:



Символ с индексом 2 не входит в диапазон.

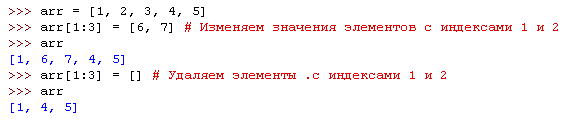
А теперь получим последний элемент:



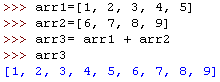
И, наконец, выведем фрагмент от второго элемента до четвертого включительно:



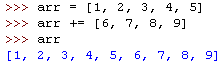
С помощью среза можно изменить фрагмент списка. Если срезу присвоить пустой список, то элементы, попавшие в срез, будут удалены:



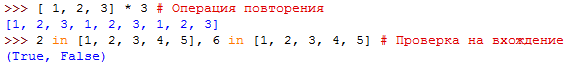
Соединить два списка в один список позволяет оператор +. Результатом объединения будет новый список:



Вместо оператора + можно использовать оператор +=. Следует учитывать, что в этом случае элементы добавляются в текущий список:



Кроме рассмотренных операций, списки поддерживают операцию повторения и проверку на вхождение. Повторить список указанное количество раз можно с помощью оператора \*, а выполнить проверку на вхождение элемента в список позволяет оператор in:



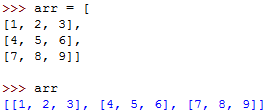
**8.3. Многомерные списки**

Любой элемент списка может содержать объект произвольнаго типа. Например, элемент списка может быть числом, строкой, списком, кортежем, словарем и т. д. Создать вложенный список можно, например, так:



Как вы уже знаете, выражение внутри скобок может располагаться на нескольких строках.

Следовательно, предыдущий пример можно записать иначе:



Чтобы получить значение элемента во вложенном списке, следует указать два индекса:



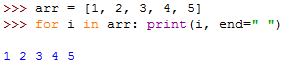
Элементы вложенного списка также могут иметь элементы произвольнаго типа. Количество вложений не ограничено. Таким образом, мы может создать объект любой степени сложности.

В этом случае для доступа к элементам указывается несколько индексов подряд. Примеры:

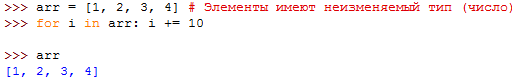


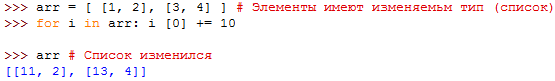
**8.4. Перебор элементов списка**

Перебрать все элементы списка можно с помощью цикла **for**:



Следует заметить, что переменную i внутри цикла можно изменить, но если она ссылается на неизменяемый тип данных (например, число или строку), то это не отразится на исходном списке:



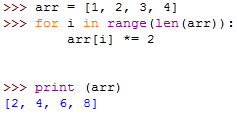


Чтобы получить доступ к каждому элементу, можно, например, воспользоваться функцией **range ()** для генерации индексов. Функция возвращает объект, поддерживающий итерации.

С помощью этого объекта внутри цикла **for** можно получить текущий индекс. Функция **range ()** имеет следующий формат:

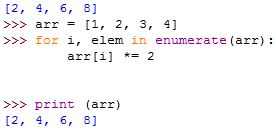
**range ( [<Начало>, ] <Конец> [, <Шаг>] )**

Первый параметр задает начальное значение. Если параметр <**Начало**> не указан, то по умолчанию используется значение 0. Во втором nараметре указывается конечное значение. Следует заметить, что это значение не входит в возвращаемый диаnазон значений. Если параметр <Шаг> не указан, то исnользуется значение 1. В качестве примера умножим каждый элемент сnиска на 2:

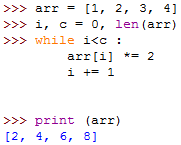


Можно также воспользоваться функцией **enumerate (<Объект> [, start=0])**, которая на каждой итерации цикла **for** возвращает кортеж из индекса и значения текущего элемента сnиска.

Умножим каждый элемент списка на 2:

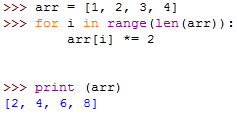


Кроме того, перебрать элементы можно с помощью цикла **while**. Но в этом случае следует помнить, что цикл **while** работает медленнее цикла **for**. В качестве примера умножим каждый элемент списка на 2, используя цикл **while**:

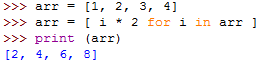


**8.5. Генераторы списков и выражения-генераторы**

В предыдущем разделе мы изменяли элементы списка следующим образом:

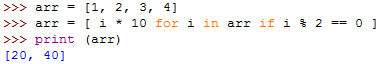


С помощью генераторов списков тот же самый код можно записать более компактно. Помимо компактного отображения генераторы списков работают быстрее цикла **for**. Однако вместо изменения исходного сnиска возвращается новый сnисок:

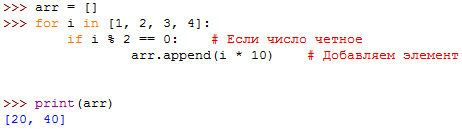


Как видно из примера, мы поместили цикл **for** внутри квадратных скобок, а также изменили порядок следования параметров: инструкция, выполняемая внутри цикла, находится перед циклом. Обратите внимание на то, что выражение внутри цикла не содержит оператора присваивания. На каждой итерации цикла будет генерироваться новый элемент, которому неявным образом присваивается результат выполнения выражения внутри цикла. В итоге будет создан новый список, содержащий измененные значения элементов исходного списка.

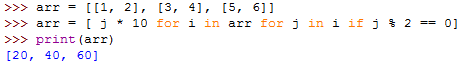
Генераторы списков могут иметь сложную структуру. Например, состоять из нескольких вложенных циклов **for** и (или) содержать оператор ветвления **if** после цикла. В качестве примера получим четные элементы списка и умножим их на 10:



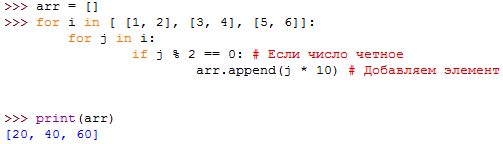
Последовательность выполнения этого кода эквивалентна последовательности выполнения следующего кода:



Усложним наш пример. Получим четные элементы вложенного списка и умножим их на 10:



Последовательность выполнения этого кода эквивалентна последовательности выполнения следующего кода:



Если выражение разместить не внутри квадратных скобок, а внутри круглых скобок, то будет возвращаться не список, а итератор. Такие конструкции называются выражениями генераторами.

В качестве примера просуммируем четные числа в списке:



**8.6. Функции mар(), zip(), filter() и reduce()**

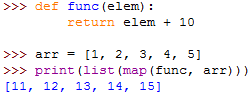
Встроенная функция **map()** позволяет применить функцию к каждому элементу последовательности.

Функция имеет следующий формат:

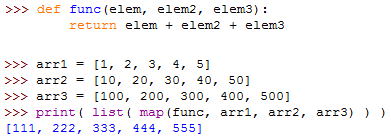
**mар(<Функция>, <Последовательность1>[, ... , <ПоследовательностьN>])**

Функция **map()** возвращает объект, nоддерживающий итерации, а не сnисок, как это было ранее в **Python 2**. Чтобы получить список, необходимо результат nередать в функцию **list()**.

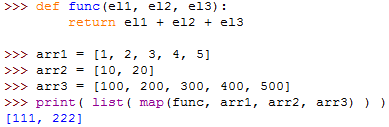
В качестве параметра <**Функция**> указывается ссылка на функцию (название функции без круглых скобок), которой будет nередаваться текущий элемент nоследовательности. Внутри функции обратного вызова необходимо вернуть новое значение. Прибавим к каждому элементу списка число 10.



Функции **map()** можно передать несколько последовательностей. В этом случае в функцию обратного вызова будут передаваться сразу несколько элементов, расположенных в последовательностях на одинаковом смещении. Просуммируем элементы трех списков.



Если количество элементов в последовательностях будет разным, то в качестве ограничения выбирается последовательность с минимальным количеством элементов:



Встроенная функция **zip()** на каждой итерации возвращает кортеж, содержащий элементы последовательностей, которые расположены на одинаковом смещении. Функция возвращает объект, поддерживающий итерации, а не список, как это было ранее в **Python 2**. Чтобы получить список, необходимо результат передать в функцию **list()**.

Формат функции:

**zip( <Последовательностьl> [, ... , <ПоследовательностьN>] )**

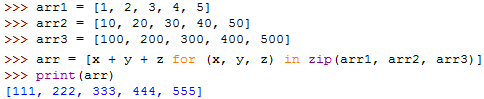
Пример:



Если количество элементов в последовательностях будет разным, то в результат поnадут только элементы, которые существуют во всех последовательностях на одинаковом смешении:



В качестве еше одного примера переделаем нашу nрограмму суммирования элементов трех списков и используем функцию **zip()** вместо функции **map()**.



Функция **filter()** позволяет выполнить проверку элементов последовательности. Формат функции:

**filter(<Функция>, <Последовательность>)**

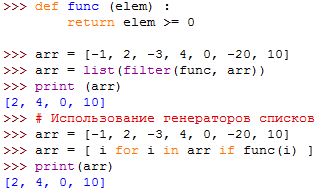
Если в первом nараметре вместо названия функции указать значение **None**, то каждый элемент последонательности будет проверен на соответствие значению **True**. Если элемент в логическом контексте возвращает значение **False**, то он не будет добавлен в возвращаемый результат. Функция возвращает объект, поддерживающий итерации, а не список или кортеж, как это было ранее в **Python 2**. Чтобы получить сnисок в версии **Python 3**, необходимо результат nередать в функцию **list ()**. Пример:



Аналогичная операция с использованием генераторов списков выглядит так:



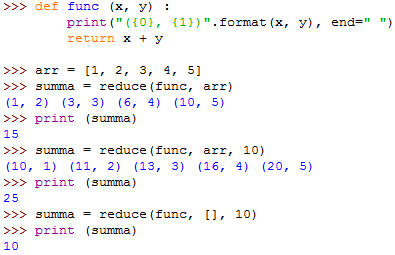
В первом параметре можно указать ссылку на функцию. В эту функцию в качестве параметра будет передаваться текущий элемент последовательности. Если элемент следует добавить в возвращаемое функцией **filter()** значение, то внутри функции обратного вызова следует вернуть значение **True**, в противном случае - значение **False**. Удалим все отрицательные значения из списка.



Функция **reduce()** из модуля **functools** применяет указанную функцию к парам элементов и накапливает результат. Имеет следующий формат:

**rеduсе(<Функция>, <Последовательность>[, <Начальное значение>])**

В функцию обратного вызова в качестве параметров будут nередаваться два элемента. Первый элемент будет содержать результат предыдущих вычислений, а второй – значение текущего элемента. Получим сумму всех элементов списка.

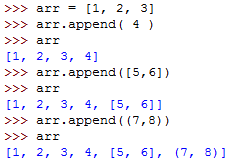


**8.7. Добавление и удаление элементов списка**

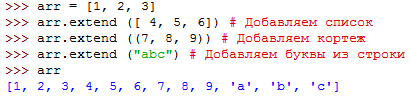
Для добавления и удаления элементов списка используются следующие методы:

* **append ( <Объект>)** - добавляет один объект в конец сnиска. Метод изменяет текущий список и ничего не возвращает.

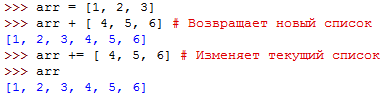
Пример:



* **extend (<Последовательность>)** - добавляет элементы последовательности в конец списка. Метод изменяет текущий список и ничего не возвращает. Пример:



Добавить несколько элементов можно с помощью операции конкатенации или оператора+=:

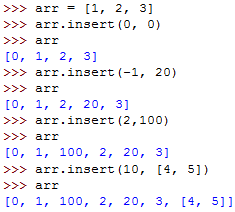


Кроме того, можно воспользоваться операцией присваивания значения срезу:

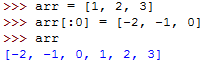


* **insert (<Индекс>, <Объект>)** - добавляет один объект в указанную позицию. Остальные элементы смещаются. Метод изменяет текущий список и ничего не возвращает.

Примеры:

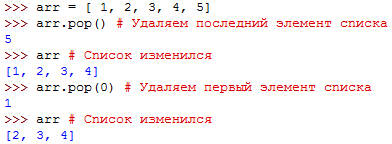


Метод **insert()** позволяет добавить только один объект. Чтобы добавить несколько объектов, можно воспользоваться операцией присваивания значения срезу. Добавим несколько элементов в начало списка:

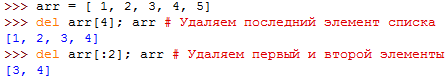


* **рор ( [·<Индекс>] )** - удаляет элемент, расположенный по указанному индексу, и возвращает eго. Если индекс не указан, то удаляет и возвращает последний элемент списка.

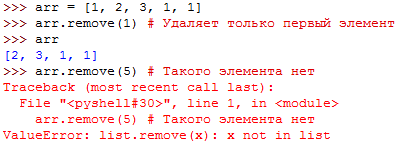
Если элемента с указанным индексом нет или список пустой, возбуждается исключение **IndexError**. Примеры:



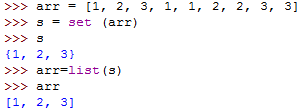
Удалить элемент списка позволяет также оператор **del**:



* **remove (<Значение>)** - удаляет первый элемент, содержащий указанное значение. Если элемент не найден, возбуждается исключение **valueError**. Метод изменяет текущий список и ничего не возвращает. Примеры:



Если необходимо удалить все повторяющиеся элементы списка, то можно список преобразовать во множество, а затем множество обратно преобразовать в список. Обратите внимание на то, что список должен содержать только неизменяемые объекты (например, числа, строки или кортежи). В противном случае возбуждается исключение **TypeError**. Пример:



**8.8. Поиск элемента в списке**

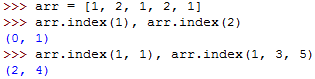
Как вы уже знаете, выполнить проверку на вхождение элемента в список позволяет оператор **in**. Если элемент входит в список, то возвращается значение **True**, в противном случае - **Fa1se**. Пример:



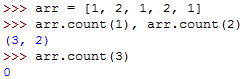
Тем не менее, оператор **in** не дает никакой информации о местонахождении элемента внутри списка. Чтобы узнать индекс элемента внутри списка, следует воспользоваться методом **index()**. Формат метода:

**indех(<Значение>[, <Начало>[, <Конец>]])**

Метод **index()** возвращает индекс элемента, имеющего указанное значение. Если значение не входит в список, то возбуждается исключение **ValueError**. Если второй и третий параметры не указаны, то поиск будет производиться с начала списка. Пример:



Узнать общее количество элементов с указанным значением позволяет метод **count (<Значение>)**. Если элемент не входит в список, то возвращается значение 0. Пример:



С помощью функций **max()** и **min()** можно узнать максимальное и минимальное значение списка соответственно. Пример:



Функция **аnу(<Последовательность>)** возвращает значение **True**, если в последовательности существует хоть один элемент, который в логическом контексте возвращает значение **True**. Если последовательность не содержит элементов, возвращается значение **False**. Пример:



Функция **all (<Последовательность>)** возвращает значение **True**, если все элементы последовательности в логическом контексте возвращают значение **True** или последовательность не содержит элементов.

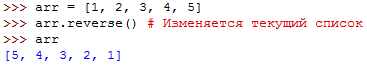
Пример:



**8.9. Первворачивание списка**

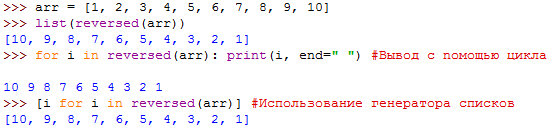
Метод **reverse()** изменяет порядок следования элементов списка на противоположный.

Метод изменяет текущий список и ничего не возвращает. Пример:



Если необходимо изменить порядок следования и· получить новый список, то следует воспользоваться

функцией **reversed (<Последовательность>)**. Функция возвращает итератор, который можно преобразовать в список с помощью функции **list()** или генератора списков:



**8.1 Генерация случайных чисел (модуль random)**

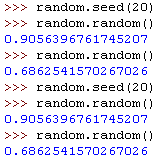
Модуль **random** позволяет генерировать случайные числа. Прежде чем использовать модуль, необходимо подключить его с помощью инструкции:

**import random**

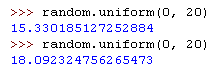
**random.random()** — возвращает псевдослучайное число от 0.0 до 1.0



**random.seed(<Параметр>)** — настраивает генератор случайных чисел на новую последовательность. По умолчанию используется системное время. Если задается значение параметра, то каждый раз при задании этого параметра будет выдаваться одна и та же последовательность чисел.



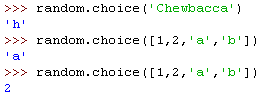
**random.uniform(<Начало>, <Конец>)** — возвращает псевдослучайное вещественное число в диапазоне от <Начало> до <Конец>:



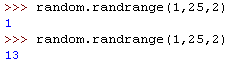
**random.randint(<Начало>, <Конец>)** — возвращает псевдослучайное целое число в диапазоне от <Начало> до <Конец>:



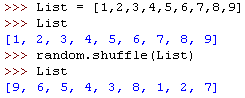
**random.choice(<Последовательность>)** — возвращает случайный элемент из любой последовательности (строки, списка, кортежа):



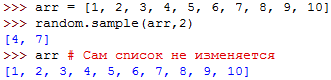
**random.randrange(<Начало>, <Конец>, <Шаг>)** — возвращает случайно выбранное число из последовательности.



**random.shuffle(<Список>)** — перемешивает последовательность (изменяется сама последовательность). Поэтому функция не работает для неизменяемых объектов.



**random.sample (<Последовательность>, <Количество элементов>)** - возвращает список из указанного количества элементов. В этот список попадут элементы из последовательности, выбранные случайным образом. В качестве последовательности можно указать любые объекты, поддерживающие итерации. Пример:

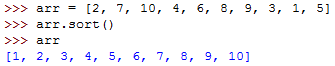


**8.11. Сортировка списка**

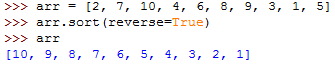
Отсортировать список позволяет метод **sort()**. Метод имеет следующий формат:

**sort([key=None] [, reverse=false])**

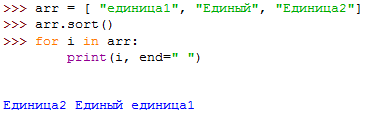
Все параметры являются необязательными. Метод изменяет текущий список и ничего не возвращает. Отсортируем список по возрастанию с параметрами по умолчанию:



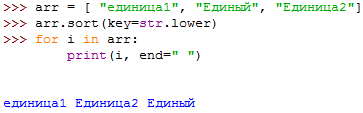
Чтобы отсортировать список по убыванию, следует в параметре **reverse** указать значение **True**:



Следует заметить, что стандартная сортировка зависит от регистра символов.



В результате мы получили неправильную сортировку, ведь Единый и Единица2 больше единица1. Чтобы регистр символов не учитывался, можно указать ссылку на функцию для изменения регистра символов в параметре **key**.

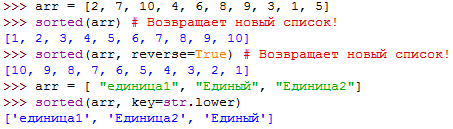


Метод **sort()** сортирует сам список и не возвращает никакого значения. В некоторых случаях необходимо получить отсортированный список, а текущий список оставить без изменений.

Дпя этого следует воспользоваться функцией **sorted()**. Функция имеет следующий формат:

**sоrtеd(<Последовательность>[, key=None] [, reverse=False])**

В первом параметре указывается список, который необходимо отсортировать. Остальные·параметры эквивалентны параметрам метода **sort()**. Пример использования функции **sorted()** приведен ниже.



**8.12. Заполнение списка числами**

Создать список, содержащий диапазон чисел, можно с помощью функции **range()**. Функция возвращает объект, поддерживающий итерации. Чтобы из этого объекта получить список, следует воспользоваться функцией **list()**. Функция **range()** имеет следующий формат:

**range([<Начало>, ] <Конец> [, <Шаг>])**

Первый параметр задает начальное значение. Если параметр <**Начало**> не указан, то по умолчанию используется значение 0. Во втором параметре указывается конечное значение. Следует заметить, что это значение не входит в возвращаемый список значений. Если параметр <**Шаг**> не указан, то используется значение **1**. В качестве примера заполним список числами от 0 до 10:



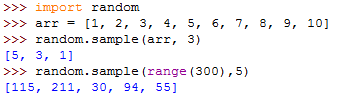
Создадим список, состоящий из диапазона чисел от 1 до 15:



Теперь изменим порядок следования чисел на противоположный:



Если необходимо получить список со случайными числами (или случайными элементами из другого списка), то следует воспользоваться функцией   
**sample(<Последовательность>, <Количество элементов>)** из модуля **random**. Пример:



**8.13. Преобразование списка в строку**

Преобразовать список в строку позволяет метод **join()**. Элементы добавляются через указанный разделитель. Формат метода:

**<Строка> = <Разделитель>.jоin(<Последовательность>)**

Пример:



Обратите внимание на то, что элементы списка должны быть строками, иначе возвращается исключение **TypeError**.

Избежать этого исключения можно с помощью выражения-генератора, внутри которого текущий элемент списка преобразуется в строку с помощью функции **str():**



Кроме того, с помощью функции **str()** можно сразу получить строковое представление списка:



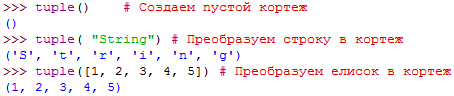
**8.14. Кортежи**

Кортежи, так же как и списки, являются упорядоченными последовательностями элементов. Кортежи во многом аналогичны спискам, но имеют одно очень важное отличие – изменить кортеж нельзя. Можно сказать, что кортеж - это список, доступный "только для чтения".

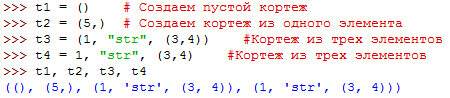
Создать кортеж можно следующими способами:

* с помощью функции **tuple([<Последовательность>])**. Функция позволяет преобразовать любую последовательность в кортеж. Если параметр не указан, то создается пустой кортеж.

Пример:

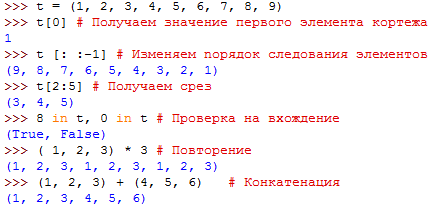


* указав все элементы через запятую внутри круглых скобок (или без скобок):



Обратите особое внимание на вторую строку примера. Чтобы создать кортеж из одного элемента, необходимо в конце указать запятую. Именно запятые формируют кортеж, а не круглые скобки. Если внутри круглых скобок нет запятых, то будет создан объект другого типа. Четвертая строка в предыдущем примере также доказывает, что не скобки формируют кортеж, а запятые. **Помните, что любое выражение в языке Python можно заключить в круглые скобки, а чтобы получить кортеж, необходимо указать запятые.**

Позиция элемента в кортеже задается индексом. Обратите внимание на то, что нумерация элементов кортежа (как и списка) начинается с 0, а не с 1. Как и все последовательности, кортежи поддерживают обращение к элементу по индексу, получение среза, конкатенацию (оператор+), повторение (оператор\*), проверку на вхождение (оператор in). Пример:



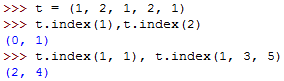
Кортежи относятся к неизменяемым типам данных. Иными словами, можно получить элемент по индексу, но изменить его нельзя:

Получить количество элементов кортежа nозволяет функция **len():**

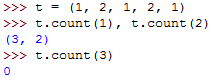


Начиная с **Python 2.6**, кортежи nоддерживают два метода:

* **index(<Значение> [, <Начало> [, <Конец>]])** - возвращает индекс элемента, имеющего указанное значение. Если значение не входит в кортеж, возбуждается исключение **ValueError**. Если второй и третий параметры не указаны, то поиск будет производиться с начала кортежа. Пример:



* **count (<Значение>)** - возвращает количество элементов с указанным значением. Если элемент не входит в кортеж, то возвращается значение 0. Пример:



Других методов у кортежей нет. Чтобы произвести операции над кортежами, следует воспользоваться встроенными функциями, nредназначенными для работы с nоследовательностями.

Все эти функции были рассмотрены при изучении списков.