Moscow Coding School

Python как первый язык

Преподаватель: Захарчук Сергей Сергеевич

Сегодня

- Повторение изученного материала
- Структура данных
- Ссылка- значение
- Словари, строки, кортежи, листы, сеты

Строковый тип

Строка – это последовательность символов с произвольным доступом. Строки в языке Python невозможно изменить – в этом случае говорят, что это immutable тип. Попытка изменить символ в определенной позиции или подстроку вызовет ошибку:

```
>>> word = ’string’
>>> word[2] = 'y'
TypeError: 'str' object does not support item assignment
```

Строковый тип

Но если очень хочется, то изменить можно, например, так:

```
>>> word = word[:3] + '!' + word[4:]
'str!ng'
```

Строковый тип

Или так:

```
>>> word = word.replace('!','e')
'streng'
```

Строковый тип

Индексы могут иметь отрицательные значения для отсчета с конца – отсчет начинается с -1:

```
>>> word[-1]
h
```

Строковый тип

Строки в питоне можно заключать как в одинарные, так и в двойные кавычки, причем кавычки одного типа могут быть произвольно вложены в кавычки другого типа:

```
>>> '123' '123’
 >>> "7'8''9"
"7'8''9"
```

Строковый тип

Строки в питоне можно заключать как в одинарные, так и в двойные кавычки, причем кавычки одного типа могут быть произвольно вложены в кавычки другого типа:

```
>>> '123' '123’
 >>> "7'8''9"
"7'8''9"
```

Строковый тип

Длинные строки можно разбивать на несколько строк с помощью обратного слеша:

```
>>> s = 'this is first word\
and this is second word'
```

Строковый тип

Большие наборы строк и целые тексты можно заключать в тройные кавычки:

```
>>> print ("""
One
Two
Three """)
```

Строковый тип

Большие наборы строк и целые тексты можно заключать в тройные кавычки:

```
>>> print ("""
One
Two
Three """)
```

Строковый тип

- Обратный слеш в строках используется для так называемой escape-последовательности.
- После слеша может идти один или несколько символов.
- В следующем примере комбинация '\n' – это новая строка, '\t' – это табуляция:

```
>>> s ='a\nb\tc'
>>> print s
a
b	c
```

Форматирование строк

Форматирование в питоне – мощный инструмент управления строками. Есть несколько подходов – стандартный и с использованием шаблонов. Для форматирования в питоновских строках используется стандартный оператор – символ %. Слева от процента указываем строку, справа – значение или список значений:

```
>>> s = 'Hello %s' % 'word'
>>> s
'Hello word’
 >>> s = 'one %s %s' % ('two','three')
>>> s
'one two three'
```

Форматирование строк

- Иногда (а точнее, довольно часто) возникают ситуации, когда нужно сделать строку, подставив в неё некоторые данные, полученные в процессе выполнения программы (пользовательский ввод, данные из файлов и т. д.). Подстановку данных можно сделать с помощью форматирования строк. Форматирование можно сделать с помощью оператора %, и метода format
- **Метод format является наиболее правильным**, но часто можно встретить программный код с форматированием строк в форме оператора %.

Форматирование строк

Если нужно преобразование числа в строку, используется числовой спецификатор – %d или %f:

```
>>> s = 'one %d %f' % (2 , 3.5)
>>> s
'one 2 3.500000’
```

```
>>> '%d %s, %d %s' % (6, 'bananas', 10, 'lemons')
'6 bananas, 10 lemons'
```

Форматирование строк

При форматировании можно указать общую ширину строки и точность для чисел, при этом число будет дополнено незначащими нулями. В следующем примере результирующая строка будет иметь длину 10 символов, на дробную часть будет отведено 5 символов:

```
>>> x = 4/3
>>> '%10.5f' % x
' 1.33333’
```

Пробелы слева можно отформатировать нулями:

```
>>> from math import pi
>>> '%015.10f' % pi
'0003.1415926536'
```

Форматирование строк

Для форматирования можно использовать другой подход – шаблоны строк, Template. В следующем примере для форматирования уже можно использовать словарь:

```
>>> from string import Template
>>> s = Template('1 $two 3 4 $five')
>>> d={}
>>> d['two']=2
>>> d['five']=5
>>> s.substitute(d)
'1 2 3 4 5'
```

Форматирование строк

Формат	Что получится
'%d', '%i', '%u'	Десятичное число.
'%o'	Число в восьмеричной системе счисления.
'%x'	Число в шестнадцатеричной системе счисления (буквы в нижнем регистре).
'%X'	Число в шестнадцатеричной системе счисления (буквы в верхнем регистре).
'%e'	Число с плавающей точкой с экспонентой (экспонента в нижнем регистре).
'%E'	Число с плавающей точкой с экспонентой (экспонента в верхнем регистре).
'%f', '%F'	Число с плавающей точкой (обычный формат).
'%g'	Число с плавающей точкой. с экспонентой (экспонента в нижнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'%G'	Число с плавающей точкой. с экспонентой (экспонента в верхнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'%c'	Символ (строка из одного символа или число - код символа).
'%r'	Строка (литерал python).
'%s'	Строка (как обычно воспринимается пользователем).
'%%'	Знак '%'.

Форматирование строк

Флаги преобразования:

Флаг	Значение
"#"	Значение будет использовать альтернативную форму.
"0"	Свободное место будет заполнено нулями.
"-"	Свободное место будет заполнено пробелами справа.
" "	Свободное место будет заполнено пробелами справа.
"+"	Свободное место будет заполнено пробелами слева.

Форматирование строк

```
>>> '%.2s' % 'Hello!'
'He'
>>> '%.*s' % (2, 'Hello!')
'He'
>>> '%-10d' % 25
'25                '
>>> '%+10f' % 25
'+25.000000'
>>> '%+10s' % 'Hello'
'          Hello'
```

Форматирование строк

Форматирование строк с помощью метода format

Если для подстановки требуется только один аргумент, то значение - сам аргумент:

```
>>> 'Hello, {}!'.format('Vasya')
'Hello, Vasya!'
```

Форматирование строк

А если несколько, то значениями будут являться все аргументы со строками подстановки (обычных или именованных):

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{}, {}, {}'.format('a', 'b', 'c')
 'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
 'c, b, a'
>>> '{2}, {1}, {0}'.format(*'abc')
 'c, b, a'
>>> '{0}{1}{0}'.format('abra', 'cad')
 'abracadabra'
>>> 'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-115.81W')
'Coordinates: 37.24N, -115.81W'
>>> coord = {'latitude': '37.24N', 'longitude': '-115.81W'}
>>> 'Coordinates: {latitude}, {longitude}'.format(**coord)
'Coordinates: 37.24N, -115.81W'
```

Форматирование строк

Однако метод format умеет большее. Вот его синтаксис:

Поле замены

"{" [имя поля] ["!" преобразование] [":" спецификация] "}"

Имя поля

arg_name ("." имя атрибута | "[" индекс "]")*

Преобразование

"r" (внутреннее представление) | "s" (человеческое представление)

Форматирование строк

Спецификация

[[fill]align][sign][#][0][width][,][.precision][type]

Заполнитель

Любой символ, кроме '{' или '}'

Выравнивание ::= "<" | ">" | "=" | "^"

Знак ::= "+" | "-" | " "

Ширина и**Точность**

Любое число

Тип ::= "b" | "c" | "d" | "e" | "E" | "f" |

Форматирование строк

Выравнивание производится при помощи символа-заполнителя. Доступны следующие варианты выравнивания:

Флаг	Значение
'<'	Символы-заполнители будут справа (выравнивание объекта по левому краю) (по умолчанию).
'>'	выравнивание объекта по правому краю.
'='	Заполнитель будет после знака, но перед цифрами. Работает только с числовыми типами.
'^'	Выравнивание по центру.

Форматирование строк

Опция "знак" используется только для чисел и может принимать следующие значения:

Флаг	**Значение**
'+'	Знак должен быть использован для всех чисел.
'-'	'-' для отрицательных, ничего для положительных.
'Пробел'	'-' для отрицательных, пробел для положительных.

Форматирование строк

Поле "тип" может принимать следующие значения:

Тип	Значение
'd', 'i', 'u'	Десятичное число.
'o'	Число в восьмеричной системе счисления.
'x'	Число в шестнадцатеричной системе счисления (буквы в нижнем регистре).
'X'	Число в шестнадцатеричной системе счисления (буквы в верхнем регистре).
'e'	Число с плавающей точкой с экспонентой (экспонента в нижнем регистре).
'E'	Число с плавающей точкой с экспонентой (экспонента в верхнем регистре).
'f', 'F'	Число с плавающей точкой (обычный формат).
'g'	Число с плавающей точкой. с экспонентой (экспонента в нижнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'G'	Число с плавающей точкой. с экспонентой (экспонента в верхнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'c'	Символ (строка из одного символа или число - код символа).
's'	Строка.
'%'	Число умножается на 100, отображается число с плавающей точкой, а за ним знак %.

Форматирование строк

Несколько примеров:

```
>>> coord = (3, 5)
>>> 'X: {0[0]}; Y: {0[1]}'.format(coord)
'X: 3; Y: 5’
 >>> "repr() shows quotes: {!r}; str() doesn't: {!s}".format('test1', 'test2')
"repr() shows quotes: 'test1'; str() doesn't: test2"
>>> '{:<30}'.format('left aligned')
'left aligned
```

Форматирование строк

Несколько примеров:

```
>>> coord = (3, 5)
>>> 'X: {0[0]}; Y: {0[1]}'.format(coord)
'X: 3; Y: 5’
 >>> "repr() shows quotes: {!r}; str() doesn't: {!s}".format('test1', 'test2')
"repr() shows quotes: 'test1'; str() doesn't: test2"
>>> '{:<30}'.format('left aligned')
'left aligned
```

Форматирование строк

Например:

```
>>> "Units destroyed: {players[0]}".format(players = [1, 2, 3])
'Units destroyed: 1'
>>> "Units destroyed: {players[0]!r}".format(players = ['1', '2', '3']) "Units destroyed: '1'”
>>> "int: {0:d}; hex: {0:#x}; oct: {0:#o}; bin: {0:#b}".format(42)
'int: 42; hex: 0x2a; oct: 0o52; bin: 0b101010’
>>> points = 19.5
 >>> total = 22
 >>> 'Correct answers: {:.2%}'.format(points/total)
'Correct answers: 88.64%'
```

Методы строк

Строки обладают большим набором разнообразных методов. Наиболее популярные из них:

- find – находит подстроку в строке – возвращает позицию вхождения строки, либо -1:

```
>>> s = 'The find method finds a substring’
 >>> s.find('find')
4
>>> s.find('finds')
16
>>> s.find('findsa')
-1
```

Методы строк

- join – объединяет через разделитель набор строк:

```
>>> s= ['one','two','three']
>>> sep = ','
>>> sep.join(s)
'one,two,three'
```

Методы строк

- split – это обратная функция для join, разбивает строку на последовательность:

```
>>> s = '/usr/local/bin'
>>> s.split('/')
['', 'usr', 'local', 'bin']
```

Методы строк

- replace – заменяет в строке одну подстроку на другую:

```
>>> s = 'replace method returns a string'
>>> s.replace('returns','return')
'replace method return a string'
```

Методы строк

- strip – удаляет пробелы слева и справа:

```
>>> '                this is whitespace string                '.strip()
'this is whitespace string'
```

Методы строк

- translate – в отличие от replace, может делать множественную замену. В следующем примере каждый символ '1' в исходной строке будет заменен на символ '3', а символ '2' – на символ '4' соответственно:

```
>>> intab = b"aeiou"
>>> outtab = b"12345"
>>> trantab = bytes.maketrans(intab, outtab)
>>> strg = b"this is string example....wow!!!"
>>> print(strg.translate(trantab));
b'th3s 3s str3ng 2x1mpl2....w4w!!!'
```

OR

```
>>>"this is string
example....wow!!!".translate(bytes.maketrans(b"aeiou",b"12345"))
```

Методы строк

Для конверсии различных типов в строковый используются функции str, int, ord, chr:

str – конвертирует число в строку;

int – конвертирует строку в число;

ord – возвращает значение байта;

chr – конвертирует число в символ.

Структура данных

Структура данных — программная единица, позволяющая хранить и обрабатывать множество однотипных или логически связанных данных, предоставляющая набор функций для манипуляции данными (интерфейс структуры данных).

На повестке дня

- Что такое список.
- Операции со списками.
- Встроенные функции.
- Стек и очередь.
- Кортежи (Tuple).
- Сеты (Set).
- Словари (Dict)

Что такое список

Для группировки множества элементов в питоне используется список **list**, который может быть записан как индексированная последовательность значений, разделенных запятыми, заключенная в квадратные скобки. Списки имеют произвольную вложенность, т.е. могут включать в себя любые вложенные списки. Физически список представляет собой массив указателей (адресов) на его элементы. С точки зрения производительности (**performance**) списки имеют следующие особенности.

Что такое список

- Время доступа к элементу есть величина постоянная и не зависит от размера списка.
- Время на добавление одного элемента в конец списка есть величина постоянная.
- Время на вставку зависит от того, сколько элементов находится справа от него, т.е. чем ближе элемент к концу списка, тем быстрее идет его вставка.
- Удаление элемента происходит так же, как и в пункте 3.
- Время, необходимое на реверс списка, пропорционально его размеру — O(n).
- Время, необходимое на сортировку, зависит логарифмически от размера списка.
- Элементы списка не обязательно должны быть одного типа. Приведем вариант статического определения списка:

Что такое список

- Время доступа к элементу есть величина постоянная и не зависит от размера списка.
- Время на добавление одного элемента в конец списка есть величина постоянная.
- Время на вставку зависит от того, сколько элементов находится справа от него, т.е. чем ближе элемент к концу списка, тем быстрее идет его вставка.
- Удаление элемента происходит так же, как и в пункте 3.
- Время, необходимое на реверс списка, пропорционально его размеру — O(n).
- Время, необходимое на сортировку, зависит логарифмически от размера списка.

Что такое список

Элементы списка не обязательно должны быть одного типа. Приведем вариант статического определения списка:

```
>>> lst = ['spam', 'drums', 100, 1234]
```

Как и для строк, для списков нумерация индексов начинается с нуля. Для списка можно получить срез, объединить несколько списков и так далее:

```
>>> lst[1:3]
['drums', 100]
```

Что такое список

Можно менять как отдельные элементы списка, так и диапазон:

```
>>> lst[3] = 'piano'
>>> lst[0:2] = [1,2]
>>> lst
[1, 2, 100, 'piano']
```

Что такое список

Вставка:

```
>>> lst[1:1] = ['guitar','microphone']
>>> lst
[1, 'guitar', 'microphone', 2, 100, 'piano']
```

Что такое список

Можно сделать выборку из списка с определенной частотой:

```
>>> numbers = [1,2,3,4,5,6,7,8,9,0]
>>> numbers[::4]
[1, 5, 9]
```

Операции со списками

К операциям мы относим:

- копирование списка;
- сложение и умножение списков;
- итерацию — проход в цикле по элементам списка;
- конструктор списков (list comprehension);
- распаковку списка — sequence unpacking.

Операции со списками

Создание копии списка.

L1 = L2[:] — создание второй копии списка.

Здесь создается вторая копия обьекта.

L1 = list(L2) — тоже создание второй копии списка.

L1 = L2 — создание второй ссылки, а не копии. 3-й вариант показывает, что создаются две ссылки на один и тот же обьект, а не две копии.

Операции со списками

Сложение или конкатенация списков:

L1 + L2

Умножение, или повтор списков:

L1 * 2

Итерацию списков в питоне можно делать несколькими различными способами:

- простая итерация списка: for x in L:
- сортированная итерация: for x in sorted(L):
- уникальная итерация: for x in set(L):
- итерация в обратном порядке: for x in reversed(L):
- исключающая итерация — например, вывести элементы 1-го списка, которых нет во 2-м списке: for item in set(L).difference(L2)

Отступление. Циклы

Цикл while

```
>>> i = 5
>>> while i < 15:
 ... print(i)
... i = i + 2
...
```

Отступление. Циклы

Цикл for

```
>>> for i in 'hello world':
... print(i * 2, end='')
...
hheelllloo wwoorrlldd
```

Отступление. Циклы

Оператор continue

Оператор continue начинает следующий проход цикла, минуя оставшееся тело цикла (for или while)

```
>>> for i in 'hello world':
        ... if i == 'o':
                ... continue
        ... print(i * 2, end='')
...
hheellll wwrrlldd
```

Отступление. Циклы

Оператор break

Оператор break досрочно прерывает цикл.

```
>>> for i in 'hello world':
        ... if i == 'o':
                ... break
        ... print(i * 2, end='')
...
hheellll
```

Отступление. Циклы

Else

Слово else, примененное в цикле for или while, проверяет, был ли произведен выход из цикла инструкцией break, или же "естественным" образом. Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошел без помощи break.

```
>>> for i in 'hello world':
        ... if i == 'a':
                ... break
... else:
        ... print('Буквы a в строке нет')
...
Буквы a в строке нет
```

Отступление. Циклы

Вот с циклами и всё. Это просто =)

Операции со списками

Сложение или конкатенация списков:

L1 + L2

Умножение, или повтор списков:

L1 * 2

Итерацию списков в питоне можно делать несколькими различными способами:

- простая итерация списка: for x in L:
- сортированная итерация: for x in sorted(L):
- уникальная итерация: for x in set(L):
- итерация в обратном порядке: for x in reversed(L):
- исключающая итерация — например, вывести элементы 1-го списка, которых нет во 2-м списке: for item in set(L).difference(L2)

Операции со списками

Для генерации списков, кроме статической формы, можно использовать конструктор списков — list comprehension — цикл внутри квадратных скобок — на примере списка квадратов первых 10 натуральных чисел:

```
>>> a = [ i*i for i in range(1,10)]

>>> a

[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Операции со списками

Конструктор может быть условным — найдем квадраты четных натуральных чисел:

```
>>> a = [ i*i for i in range(1,10) if i % 2 == 0]
>>> a
[4, 16, 36, 64]
```

Операции со списками

Операция Sequence unpacking — присваивание списку переменных списка значений:

```
a, b = [1,2]
```

Операции со списками

Списки в Python - упорядоченные изменяемые коллекции объектов произвольных типов (почти как массив, но типы могут отличаться)!

```
>>> s = [] # Пустой список
>>> l = ['s', 'p', ['isok'], 2]
>>> s []
>>> l
['s', 'p', ['isok'], 2]
```

Встроенные функции

Метод	Что делает
list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, добавляя в конец все элементы списка L
list.insert(i, x)	Вставляет на i-ый элемент значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует
list.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
list.count(x)	Возвращает количество элементов со значением x
list.sort([key=функция])	Сортирует список на основе функции
list.reverse()	Разворачивает список
list.copy()	Поверхностная копия списка
list.clear()	Очищает список

Встроенные функции

Запомним, что методы списков, в отличие от строковых методов, изменяют сам список, а потому результат выполнения не нужно записывать в эту переменную.

```
>>> l = [1, 2, 3, 5, 7]
>>> l.sort()
>>> l
[1, 2, 3, 5, 7]
>>> l = l.sort()
>>> print(l)
None
```

Встроенные функции

Добавлять можно как одинарные элементы, так и набор элементов. Списки могут быть вложенными — вложенный список добавим в конец с помощью append():

```
>>> lst = [1, 'guitar', 'microphone', 2, 100, 'piano']
>>> lst2 = ['sintezator','drums']
>>> lst.append(lst2)
>>> lst
[1, 'guitar', 'microphone', 2, 100, 'piano', ['sintezator', 'drums']]
```

Встроенные функции

Элемент можно добавить в произвольную позицию списка с помощью метода insert:

```
>>> lst.insert(0,'vocal')
>>> lst
['vocal', 1, 'guitar', 'microphone', 2, 100, 'piano', ['sintezator', 'drums']]
```

Встроенные функции

Для проверки, является ли элемент членом списка, есть оператор in:

```
>>> 2 in lst
True
>>> 10 in lst
False
```

Встроенные функции

index() — взять элемент списка по индексу:

```
>>> lst.index('guitar')
2
```

Встроенные функции

count() — подсчет числа повторов какого-то элемента:

```
>>> lst.count('vocal')
1
```

Встроенные функции

remove() — удаление конкретного элемента:

```
>>> lst.remove(100)
>>> lst
['vocal', 1, 'guitar', 'microphone', 2, 'piano', ['sintezator', 'drums']]
```

del — удаление по индексу:

```
>>del lst[1]
```

При удалении нужно помнить о том, что нельзя одновременно делать итерацию по списку — последствия будут непредсказуемы.

Встроенные функции

sort() — сортировка списка:

```
>>> lst.sort()
>>> lst
[1, 2, ['sintezator', 'drums'], 'guitar', 'microphone', 'piano', 'vocal']
```

Встроенные функции

reverse() — реверс списка:

```
>>> lst.reverse()
>>> lst
['vocal', 'piano', 'microphone', 'guitar', ['sintezator', 'drums'], 2, 1]
```

Встроенные функции

pop() — извлечение элемента из списка, функция без параметра удаляет по умолчанию последний элемент списка, в качестве параметра можно поставить произвольный индекс:

```
>>> lst.pop()
>>> lst
['vocal', 'piano', 'microphone', 'guitar', ['sintezator', 'drums'], 2]
```

Встроенные функции

len() — длина списка:

```
>>> len(lst)
6
```

max() — максимальный элемент списка:

```
>>> max(lst)
'vocal’
```

min() — минимальный элемент списка:

```
>>> min(lst)
2
```

extend() — аналогичен append(), добавляет последовательность элементов:

```
>>> lst.extend([3,4])
>>> lst
['vocal', 'piano', 'microphone', 'guitar', ['sintezator', 'drums'], 2, 3, 4]
```

Стек и очереди

Список можно использовать как стек — когда последний добавленный элемент извлекается первым (LIFO, last-in, first-out). Для извлечения элемента с вершины стека есть метод pop():

```
>>> stack = [1,2,3,4,5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack.pop()
>>> stack
[1, 2, 3, 4, 5, 6]
```

Стек и очереди

```
>>> queue = ['rock','in','roll']
>>> queue.append('alive')
>>> queue.pop(0)
>>> queue ['in', 'roll', 'alive']
```

Кортежи (Tuple)

Список так же может быть неизменяемым (immutable), как и строка, в этом случае он называется кортеж (tuple). Кортеж использует меньше памяти, чем список. Кортеж вместо квадратных скобок использует круглые (хотя можно и совсем без скобок). Кортеж не допускает изменений, в него нельзя добавить новый элемент, хотя он может содержать объекты, которые можно изменить:

```
>>> t = 1,[2,3]
>>> t
(1, [2, 3])
>>> t[1] = 2
TypeError: 'tuple' object does not support item assignment
>>> t[1].append(4)
>>> t
(1, [2, 3, 4])
```

Кортежи (Tuple)

Функция tuple() берет в качестве аргумента строку или список и превращает его в кортеж:

```
>>> tuple('abc')
('a', 'b', 'c')
```

Сеты (Set)

Сеты — неотсортированная коллекция уникальных элементов. Сеты поддерживают итерацию, добавление и удаление объектов и т.д. Индексация и срезы в сетах не поддерживаются. Сгенерировать сет можно с помощью функции:

```
>>> s = set('abcde')
>>> s
set(['a', 'c', 'b', 'e', 'd'])
>>> s2 = set('aghij')
>>> s2
set(['a', 'h', 'j', 'g', 'f'])
```

Сеты (Set)

Над сетами можно выполнять разные операции, например:

- вычитание:

```
>>> s3 = s - s2
>>> s3
set(['c', 'b', 'e', 'd'])
```

- сложение:

```
>>> s3 = s | s2
>>> s3
set(['a', 'c', 'b', 'e', 'd', 'g', 'i', 'h', 'j'])
```

- пересечение:

```
>>> s3 = s & s2
>>> s3
set(['a'])
```

Сеты (Set)

Сеты имеют встроенные функции:

- add() — добавление элемента:

```
>>> s.add(6)
>>> s
set(['a', 'c', 'b', 'e', 'd', 6])
```

- remove() — удаление элемента:

```
>>> s.remove('a')
>>> s
set(['c', 'b', 'e', 'd', 6])
```

- Итерация:

```
>>> for item in s:print (item)
c
b
e
d
6
```

Сеты (Set)

Сеты можно использовать для фильтрации дублей в коллекциях. Для этого коллекцию нужно сконвертировать в сет, а потом обратно:

```
>>> L = [1,2,3,4,1,2,6,7]
>>> set(L)
set([1, 2, 3, 4, 6, 7])
>>> L = list(set(L))
>>> L
[1, 2, 3, 4, 6, 7]
```

Сеты (Set)

Сеты можно использовать для работы с большими наборами данных:

Допустим, у нас имеются базы данных программистов и менеджеров:

```
>>> programmers = set(['ivanov','petrov','sidorov'])
>>> managers = set(['ivanov','moxov','goroxov'])
```

Сеты (Set)

Найти тех, кто одновременно и программист, и менеджер:

Сеты (Set)

```
>>> programmers & managers
set(['ivanov'])
```

Сеты (Set)

Найти всех программистов и менеджеров:

Сеты (Set)

```
>>> programmers | managers
set(['ivanov', 'petrov', 'sidorov', 'goroxov', 'moxov'])
```

Сеты (Set)

```
>>> programmers | managers
set(['ivanov', 'petrov', 'sidorov', 'goroxov', 'moxov'])
```

Сеты (Set)

Найти программистов, которые не менеджеры:

Сеты (Set)

Найти программистов, которые не менеджеры:

```
>>> programmers - managers
set(['petrov', 'sidorov'])
```

Сеты (Set)

С множествами можно выполнять множество операций: находить объединение, пересечение...

- len(s) - число элементов в множестве (размер множества).
- x in s - принадлежит ли x множеству s.
- **set.isdisjoint**(other) - истина, если set и other не имеют общих элементов.
- **set == other** - все элементы set принадлежат other, все элементы other принадлежат set.
- **set.issubset**(other) или **set <= other** - все элементы set принадлежат other.
- **set.issuperset**(other) или **set >= other** - аналогично.
- **set.union**(other, ...) или **set | other | ...** - объединение нескольких множеств.
- **set.intersection**(other, ...) или **set & other & ...** - пересечение.
- **set.difference**(other, ...) или **set - other - ...** - множество из всех элементов set, не принадлежащие ни одному из other.
- **set.symmetric_difference**(other); **set ^ other** - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
- **set.copy**() - копия множества.

Сеты (Set)

И операции, непосредственно изменяющие множество:

- **set.update**(other, ...); set |= other | ... - объединение.
- **set.intersection_update**(other, ...); set &= other & ... - пересечение.
- **set.difference_update**(other, ...); set -= other | ... - вычитание.
- **set.symmetric_difference_update**(other); set ^= other - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
- **set.add**(elem) - добавляет элемент в множество.
- **set.remove**(elem) - удаляет элемент из множества. KeyError, если такого элемента не существует.
- **set.discard**(elem) - удаляет элемент, если он находится в множестве.
- **set.pop**() - удаляет первый элемент из множества. Так как множества не упорядочены, нельзя точно сказать, какой элемент будет первым.
- **set.clear**() - очистка множества.

Сеты (frozenset)

Единственное отличие set от frozenset заключается в том, что set - изменяемый тип данных, а frozenset - нет.

Примерно похожая ситуация с списками и кортежами.

```
>>> a = set('qwerty')
>>> b = frozenset('qwerty')
>>> a == b
True
>>> True
True
>>> type(a - b)
<class 'set'>
>>> type(a | b)
 <class 'set'>
 >>> a.add(1)
>>> b.add(1)
Traceback (most recent call last):
                File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
```

Словари (dict)

Словари в Python - неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами.

Чтобы работать со словарём, его нужно создать. Создать его можно несколькими способами. Во-первых, с помощью литерала:

```
>>> d = {}
>>> d
{}
>>> d = {'dict': 1, 'dictionary': 2}
>>> d
{'dict': 1, 'dictionary': 2}
```

Словари (dict)

Во-вторых, с помощью функции **dict**:

```
>>> d = dict(short='dict', long='dictionary')
>>> d
{'short': 'dict', 'long': 'dictionary'}
>>> d = dict([(1, 1), (2, 4)])
>>> d
{1: 1, 2: 4}
```

Словари (dict)

В-третьих, с помощью метода fromkeys:

```
>>> d = dict.fromkeys(['a', 'b'])
>>> d {'a': None, 'b': None}
>>> d = dict.fromkeys(['a', 'b'], 100)
>>> d
{'a': 100, 'b': 100}
```

Словари (dict)

В-четвертых, с помощью генераторов словарей, которые очень похожи на генераторы списков

```
>>> d = {a: a ** 2 for a in range(7)}
 >>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

Словари (dict)

Теперь попробуем добавить записей в словарь и извлечь значения ключей:

```
>>> d = {1: 2, 2: 4, 3: 9}
>>> d[1]
2
>>> d[4] = 4 ** 2
>>> d {1: 2, 2: 4, 3: 9, 4: 16}
>>> d['1']
Traceback (most recent call last):
        File "", line 1, in d['1']
KeyError: '1'
```

Словари (dict)

- Присвоение по новому ключу расширяет словарь
- Присвоение по существующему ключу перезаписывает его
- Попытка извлечения несуществующего ключа порождает исключение

Словари (dict)

Методы словарей

dict.clear() - очищает словарь.
dict.copy() - возвращает копию словаря.
classmethod **dict.fromkeys**(seq[, value]) - создает словарь с ключами из seq и значением value (по умолчанию None).
dict.get(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).
dict.items() - возвращает пары (ключ, значение).
dict.keys() - возвращает ключи в словаре.
dict.pop(key[, default]) - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).
dict.popitem() - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.
dict.setdefault(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ с значением default (по умолчанию None).
dict.update([other]) - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).
dict.values() - возвращает значения в словаре.