Лекция 4: Форматы данных

Автор: Сергей Вячеславович Макрушин e-mail: SVMakrushin@fa.ru (mailto:SVMakrushin@fa.ru)

Финансовый универсиет, 2020 г.

При подготовке лекции использованы материалы:

- Документация к рассмотренным пакетам

V 0.2 20.09.2020

Разделы:

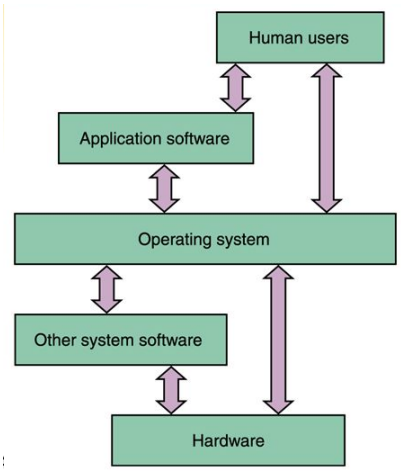
In [1]:

```
# загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v1.css")
HTML(html.read().decode('utf-8'))
```

Out[1]:

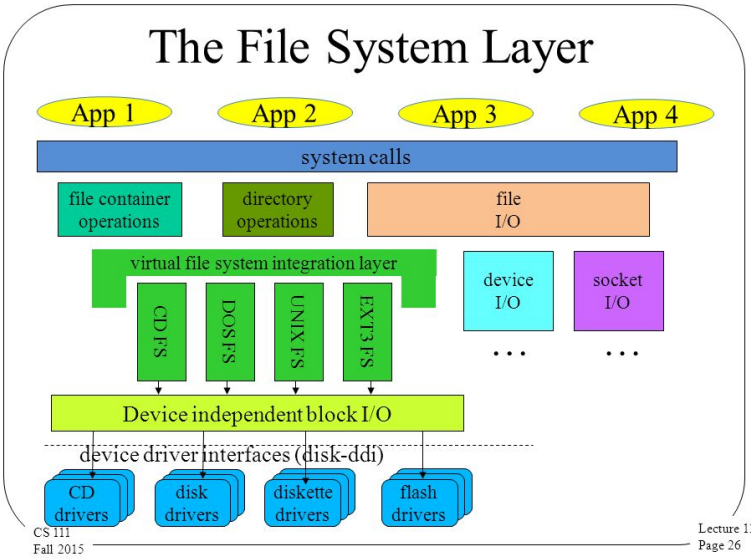
Файлы

Файл - именованная область данных на носителе информации. Работа с файлами реализуется средствами операционных систем.



Роль операционной системы

- Операционная система предоставляет приложениям набор функций и структур (API) для работы с файлами.
- Использования API абстрагирует пользователя (программу) от специфики хранения файла:
  - Существует ли файл как **объект файловой системы** или является, например, **устройством ввода-вывода**
  - В какой **файловой системе** хранится файл: зачастую операционная система позволяет работать с несколькими видами файловых систем, которые предоставляют единый базовый API для прикладной работы с файлами.
  - На каком **физическом носителе** хранится файл, например: файл может храниться на жестком диске или в оперативной памяти компьютера или на удаленном компьютере.

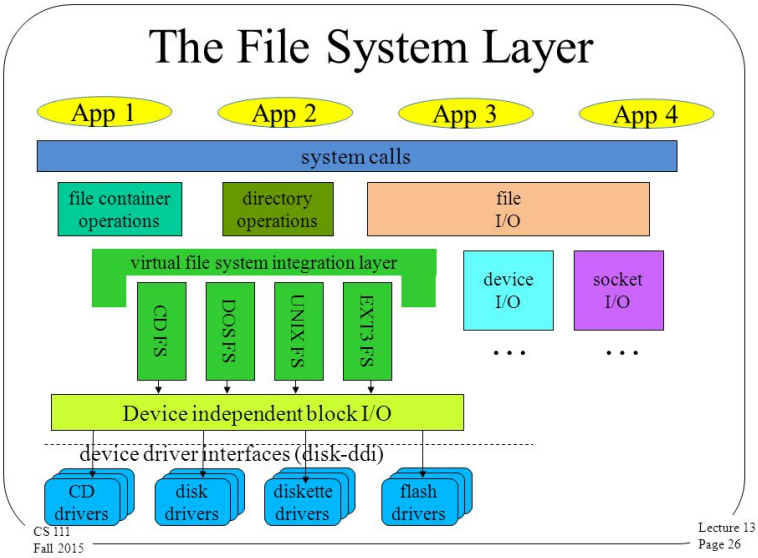


Уровни взаимодействия с файлами

- **Файловая система** - порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах. Файловая система связывает носитель информации с одной стороны и API для доступа к файлам — с другой.
- Файловая система не обязательно напрямую связана с физическим носителем информации.
  - Существуют **виртуальные файловые системы**
  - Существуют **сетевые файловые системы**, которые являются лишь способом доступа к файлам, находящимся на удалённом компьютере.
- Имеются протоколы передачи данных, которые позволяют передавать файлы по сети, в частности:
  - **FTP (File Transfer Protocol)** - протокол передачи файлов со специального файлового сервера на компьютер пользователя. FTP дает возможность абоненту обмениваться двоичными и текстовыми файлами с любым компьютером сети. Установив связь с удаленным компьютером,

пользователь может скопировать файл с удаленного компьютера на свой или скопировать файл со своего компьютера на удаленный.

- **HTTP (Hyper Text Transfer Protocol)** - это протокол передачи гипертекста. Протокол HTTP используется при пересылке Web-страниц между компьютерами, подключенными к одной сети.
- **WebDav (Web Distributed Authoring and Versioning) или просто DAV** — набор расширений и дополнений к протоколу HTTP, поддерживающих совместную работу пользователей над редактированием файлов и управление файлами на удаленных веб-серверах. Сегодня DAV применяется в качестве сетевой файловой системы, эффективной для работы в Интернете и способной обрабатывать файлы целиком, поддерживая хорошую производительность работы в условиях окружения с высокой временной задержкой передачи информации.



Стек технологий Python для обработки данных и научных расчетов

In [6]:

```
#TODO: разделители (конец строки), кодировки
```

In [7]:

```
#работа с бинарным файлом
```

Форматы файлов

Формат файла - это стандартный способ организации информации для хранения в компьютерном файле. На уровне операционной системы файл, это всего лишь контейнер для данных, способ организации данных внутри файла определяется уже форматом файла, использованным для хранения информации.

- Формат опредяет как данные кодируются в байты и как байты организуются в одномерный массив, в виде которого они хранятся в файле.
- Функционал по работе с файлами разных форматов (например их созданию из специализированных данных и чтению и использованию данных из них) реализуют специализированные программы, работающие поверх фйлового API операционной системы.
- Информация о формате файла является одним из видов метаданных.

Форматы файлов можно разделить по **способу организаци хранения данных** на:

- **Текстовые файлы** - файл, содержащий текстовые данные, представленные как оследовательность символов (в основном печатных знаков, принадлежащих к определенному набору символов - кодовой таблице (кодировке)). Эти символы обычно сгруппированы в строки (lines, rows). В современных системах строки разделяются разделителями строк, иногда конец текстового файла также отмечается одним или более специальными знаками.
  - **+** Универсальность - текстовый файл может быть прочитан на любой системе или ОС. При этом распространенной проблемой является определение **корректной кодировки файла**.
  - **+** Удобнство интерпретации файла - данные записанные в текстовом формате часто оформлены как "самозадокументированные", т.е. могут интерпретироваться человеком без дополнительных спецификаций.
  - **+** Удобнство работы с файлом - формат текстового файла крайне прост и его можно просматривать и изменять текстовым редактором - программой, доступной для любой современной ОС.
  - **+** Наличие универсальных инструментов - в частности, многие системы управления версиями рассчитаны на текстовые файлы и могут выполнять с ними множество полезных операций, например находить разницу между двумя файлами, в то время, как с двоичными файлами могут работать только как с единым целым.
  - **+** Устойчивость — каждое слово и символ в таком файле самодостаточны и, если случится повреждение байтов в таком файле, то обычно можно восстановить данные или продолжить обработку остального содержимого. У сжатых или двоичных файлов повреждение нескольких байтов может сделать файл совершенно невосстановимым.
  - **-** Низкая эффективность хранения - у больших несжатых текстовых файлов низкая информационная энтропия - эти файлы занимают больше места, нежели минимально необходимо. При этом данная избыточность определяет повышенную устойчивость данных к повреждениям.
- **Двоичные (бинарные) файлы** (binary file) - в узком смысле "не текстовые файлы". Файлы, байты в которых интерпретируются не как текст (при этом они могут включать фрагменты текста).
  - **+** Двоичные файлы сохраняют информацию аналогично представлении информации в памяти компьютера во время работы программы, что позволяет выполнять запись и чтение файла не выполняя никаких преобразований, что существенно ускоряет процесс ввода/вывода.
  - **+** Обычно хранение информации (в частности числовых массивов) в двоичном формате намного компактнее, что уменьшает требования к объему хранилища и увеличивает скорость ввода/вывода.
  - **±** Интерпретировать неизвестный двоичный формат намного сложнее что существенно усложняет работу с ним людей не имеющих доступа к инструментам, при помощи которых файлы создавались.
  - **-** Обычно двоичные файлы намного менее переносимые. Хранение в двоичном формате часто несет специфику платформы (ОС, аппаратного обеспечения) и из-за сложности обратной разработки данные форматы требуют качественной открытой спецификации. Одна из распространенных проблем: **разница в порядке байтов или длине машинного слова** на разных платформах. Подробнее см.: https://ru.wikipedia.org/wiki/Порядок_байтов (https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%
    .

In [110]:

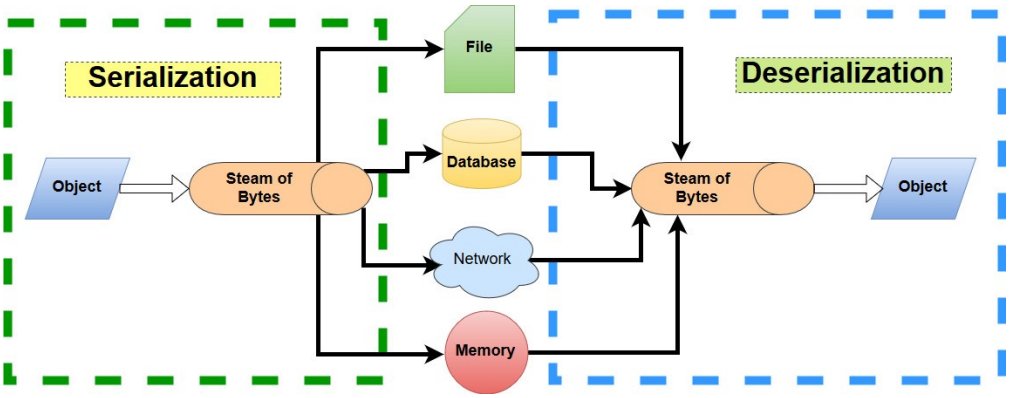
```
# Работа с двоичными и бинарными файлами в Python
```

In [ ]:

```
```

Форматы файлов можно разделить **по назначению форматов файлов**:

- **хранение данных конкретного типа** - например: PNG-файлы используются для хранения изображений.
- **контейнеры данных** - формат разработан для хранения нескольких различных типов данных. Например, формат OGG может использоваться для харанения разных типов мультимедиа информации: видое, аудио, текста (например субтитров) и метаданных.
- **сериализация данных** - перевод



Сериализация-десериализация данных

Сериализация (serialization) - процесс перевода какой-либо структуры данных в последовательность битов. Обратной к сериализации является операция **десериализации** (deserialization) — восстановление начального состояния структуры данных из битовой последовательности.

- Сериализация используется:
  - для передачи объектов по сети и для сохранения их в файлы.
  - для сохранения состояния приложения или некоторых его структур данных, хранения в файле и последующего восстановления данных.
  - Список известных форматов данных для сериализации: https://en.wikipedia.org/wiki/Comparison_of_data-serialization_formats (https://en.wikipedia.org/wiki/Comparison_of_data-serialization_formats) .
    - **XML** (eXtensible Markup Language) - расширяемый язык разметки (рекомендован W3C). XML разрабатывался как язык с простым формальным синтаксисом, **удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком**, с подчёркиванием нацеленности на использование в Интернете. Язык называется **расширяемым**, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка.
    - **JSON** (JavaScript Object Notation) - текстовый формат обмена данными, основанный на JavaScript. JSON легко читается людьми. Несмотря на происхождение от JavaScript, формат считается независимым от языка и может использоваться практически с любым языком программирования.
    - **YAML** - «дружественный» формат сериализации данных, концептуально близкий к языкам разметки, но ориентированный на удобство ввода-вывода типичных структур данных многих языков программирования.
  - Специализированные форматы, которые будут рассмотрены:
    - **pickle** - бинарный формат для сериализации объектов Python в поток байтов. Реализуется как последовательность операций для выполнения на Pickle Virtual Machine. Формат

определяется прежде всего реализацией Pickle Virtual Machine.
    - **Apache Parquet** - открытый формат хранения данных для экосистемы Apache Hadoop. Он обеспечивает эффективные схемы сжатия и кодирования данных с повышенной производительностью для обработки больших объемов сложных данных.

In [8]:

```
# Вторая часть: CSV, numpy, Parquet, xlsx
```

Спецификация формата файла - подробное описание структуры файлов данного формата, то, как программы должны кодировать данные для записи в этот формат и как декодировать их при чтении. Существуют:

- "открытые" форматы файлов - имеют публично доступные спецификации, разработанные некоммерческими организациями и частными компаниями, разработчиками формата.
- "закрытые" форматы файлов - организации разрабочики считают формат файла своей коммерческой тайной (например, старый формат файлов MS Office) или не считает нужным тратить время на написание подробной спецификации.
- Обычно форматы файлов не защищены авторскими правами и "закрытые" форматы могут быть специфицированные мтодами обратной разработкой (reverse engineering)

Метаданные и и опредеделение форматаы файлов

Формат файла - это стандартный способ организации информации для хранения в компьютерном файле. На уровне операционной системы файл, это всего лишь контейнер для данных, способ организации данных внутри файла определяется уже форматом файла, использованным для хранения информации.

- Формат опредяет как данные кодируются в байты и как байты организуются в одномерный массив, в виде которого они хранятся в файле.
- Функционал по работе с файлами разных форматов (например их созданию из специализированных данных и чтению и использованию данных из них) реализуют специализированные программы, работающие поверх фйлового API операционной системы.
- Информация о формате файла является одним из видов метаданных.

Метаданные (metadata) - "данные о данных", или "информация о другой информации", или данные, относящиеся к дополнительной информации о содержимом или объекте. Можно выделить:

- описательные метаданные (descriptive metadata) - данные используемые для обнаружения или идентификации. Например: название, аннотация. К описательным метаданным можно отнести имя файла.
- структурные метаданные (structural metadata) - данные об организации данных. В частности: **формат файла**, версия формата данных.
- административные метаданные (administrative metadata) - информация которая помогает управлять ресурсом. Например: информация о правах доступа, времени с создания файла и времени последнего изменения.
- статистические метаданные (statistical metadata) - статистическая информация о данных.

На данный момент **нет единого подхода к хранению информации об формате файла** и шире: о структурных метаданных и метаданных компьютерных файлов вообще. Возможны следующие варианты и их комбинации:

- **расширение файла** - формат файла указанный в виде текстового обозначения в конце имени файла (после последней точки в имени файла). Существует большой список общеупотребимых расширений файлов: https://en.wikipedia.org/wiki/List_of_file_formats (https://en.wikipedia.org/wiki/List_of_file_formats) , при этом не все имена уникальны, а многи форматы имеют несколько альтернативных имен, например: `htm` и `html` .
- **внутренние метаданные** - хранение информации о формате файла внтури самого файла (обычно - в самом начале файла), эта область файла именуется:
  - **заголовком файла** (file header) если область больше чем несколько байт. Например HTML файлы начинаются с `<html>` или `<!DOCTYPE HTML>` .
  - **магическим числом** (magic number) если область занимает всего несколько байт. В Unix-системах распространено использование двухбайтовых (и более длинных) магических чисел в начале файла для хранения информации о типе файла, см.: https://en.wikipedia.org/wiki/List_of_file_signatures (https://en.wikipedia.org/wiki/List_of_file_signatures) .
- **внешние метаданные** - явное хранение метаданных о файле в файловой системе.
  - **+** прогрессивный и потенциально наиболее удобный вариант
  - **-** на практике данный подход испытывает большие проблемы с переносимостью между операционными системами и файловыми системами из-за отстутсвия поддержки такого функционала многими ФС и ОС и API для передчи данных.
- **MIME types** (Multipurpose Internet Mail Extensions) - стандарт, описывающий передачу различных типов данных по электронной почте, а также, в общем случае, спецификация для кодирования информации и форматирования сообщений таким образом, чтобы их можно было пересылать по Интернету.
  - MIME определяет механизмы для передачи разного рода информации внутри текстовых данных (в частности, с помощью электронной почты), а именно: текст на языках, для которых используются кодировки, отличные от ASCII, и нетекстовые данные, такие, как картинки, музыка, фильмы и программы.
  - система MIME types имеет стандартную систему идентификторов (управляемых организацией IANA), сотоящих из типа и подтипа, разделенного слешем, например: `text/html` или `image/gif` .
  - Официальный список MIME типов на сайте IANA: https://www.iana.org/assignments/media-types/media-types.xhtml (https://www.iana.org/assignments/media-types/media-types.xhtml) .
  - На данный момент MIME **является фундаментальным компонентом коммуникационных протоколов в интернете**, в частоснти, в HTTP (HyperText Transfer Protocol) серверы вставляют заголовок MIME при передачи любых данных WWW, далее этот заголовок используется клиентом (например браузером) для корректного отображения полученных данных.
  - Роль MIME types постоянно возрастает, однако они редко используются для данных, хранимых в локальных файловых системах.

In [5]:

```
# MIME type из HTTP
```

Структурирование данных

Принято делить данные на три категории:

- **Структурированные** – **Ex:** реляционная база данных
- **Слабо (полу-) структурированные** – **Ex:** веб-страница
- **Не структурированные** – **Ex:** текст на естественном языке.

Слабо структурированное данные:

- Обычно слабо структурированные данные содержат **маркеры для отделения семантических элементов** и для обеспечения частичной структуры данных в наборе данных, но не соответствуют строгой структуре отношений реляционной модели данных. Такой вид данных можно назвать **бессхемным**, а структуру — **самоописываемой**. В слабоструктурированных данных **сущности**, принадлежащие одному и тому же классу, **могут иметь разные атрибуты**, порядок атрибутов также не важен.
- Слабоструктурированные представления данных важны, т.к. для **обмена данными** между разными системами (например в web) необходимо иметь **максимально гибкий формат**.

- Часто слабо структурированные данные именуются документами. **Документ** — объект, содержащий информацию в зафиксированном виде и специально предназначенный для её передачи во времени и пространстве. Обычно документы имеют внутреннюю структуру, при этом часто не существует стандарта такой струкутуры, т.е. в этом смысле документы являются безсхемной самоописываемой информацией.

In [ ]:

```
```

Сериализация и обмен данными

- к оглавлению

Особенности, плюсы / минусы

- **pickle** - бинарный формат для сериализации объектов Python в поток байтов. Реализуется как последовательность операций для выполнения на Pickle Virtual Machine. Формат определяется прежде всего реализацией Pickle Virtual Machine.
- **JSON** (JavaScript Object Notation) - текстовый формат обмена данными, основанный на JavaScript. JSON легко читается людьми. Несмотря на происхождение от JavaScript, формат считается независимым от языка и может использоваться практически с любым языком программирования.
- **XML** (eXtensible Markup Language) - расширяемый язык разметки (рекомендован W3C). XML разрабатывался как язык с простым формальным синтаксисом, **удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком**, с подчёркиванием нацеленности на использование в Интернете. Язык называется **расширяемым**, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка.

Формат Pickle

- к оглавлению

pickle - бинарный формат для сериализации объектов Python в поток байтов. Реализуется как последовательность операций для выполнения на Pickle Virtual Machine. Формат определяется прежде всего реализацией Pickle Virtual Machine.

The core general serialization mechanism is the pickle standard library module, alluding to the database systems term pickling[27][28][29] to describe data serialization (unpickling for deserializing). Pickle uses a

simple stack-based virtual machine that records the instructions used to reconstruct the object. It is a cross-version customisable but unsafe (not secure against erroneous or malicious data) serialization format. Malformed or maliciously constructed data, may cause the deserializer to import arbitrary modules and instantiate any object.[30][31] The standard library also includes modules serializing to standard data formats: json (with built-in support for basic scalar and collection types and able to support arbitrary types via encoding and decoding hooks). plistlib (with support for both binary and XML property list formats). xdrlib (with support for the External Data Representation (XDR) standard as described in RFC 1014). Finally, it is recommended that an object's **repr** be evaluable in the right environment, making it a rough match for Common Lisp's print-object. Not all object types can be pickled automatically, especially ones that hold operating system resources like file handles, but users can register custom "reduction" and construction functions to support the pickling and unpickling of arbitrary types. Pickle was originally implemented as the pure Python pickle module, but, in versions of Python prior to 3.0, the cPickle module (also a built-in) offers improved performance (up to 1000 times faster[30]). The cPickle was adapted from the Unladen Swallow project. In Python 3, users should always import the standard version, which attempts to import the accelerated version and falls back to the pure Python version.[32]

Сохранение данных в Pickle

- Стандартное расширение для файлов: `.pickle`
- Могут встречаться расширения: `.pkl` , `.pck` , `.db`

In [2]:

```
# объект (данные) для сохранения:
dat1 = ["Строка", (12, 3)]
```

In [3]:

```
import pickle # подключаем модуль pickle
```

In [4]:

```
# Стандартное расширение дл

with open('dat1.pickle', 'wb') as f:
    pickle.dump(dat1, f)
```

Pickle хранит данные в бинарном формате:

In [5]:

```
with open('dat1.pickle') as f:
    for l in f:
        print(l)
```

```
Ђ]q(XРЎС,СЂPsPєP°q]KK]†q]e.
```

In [6]:

```
with open('dat1.pickle', 'rb') as f:
    for l in f:
        print(l)
```

```
b'\x80\x03]q\x00(X\x0c\x00\x00\x00\xd0\xa1\xd1\x82\xd1\x80\xd0\xbe\xd0\xba\xd
0\xb0q\x01K\x0cK\x03\x86q\x02e.'
```

Восстановление объектов (данных) из файла pickle:

In [7]:

```
with open('dat1.pickle', 'rb') as f:
    dat1_l = pickle.load(f)

dat1_l
```

Out[7]:

```
['Строка', (12, 3)]
```

В один файл можно сохранить сразу несколько объектов, последовательно вызывая функцию `dump()`.

In [8]:

```
dat2 = (6, 7, 8, 9, 10)
```

In [9]:

```
with open('dat2.pickle', 'wb') as f:
    pickle.dump(dat1, f)
    pickle.dump(dat2, f)
```

In [10]:

```
# Восстановление данных из файла:

with open('dat2.pickle', 'rb') as f:
    dat1_l2 = pickle.load(f)
    dat2_l2 = pickle.load(f)
dat1_l2, dat2_l2
```

Out[10]:

```
(['Строка', (12, 3)], (6, 7, 8, 9, 10))
```

Модуль `pickle` позволяет также преобразовать объект в строку байтов и восстановить объект из строки. Для этого предназначены две функции:

- `dumps(<Объект> [, <Протокол>] [, fix_imports=True])` - производит сериализацию объекта и возвращает последовательность байтов специального формата.
- `loads(<Последовательность байтов>[, fix_imports=True] [, errors="strict"])` - преобразует последовательность байтов обратно в объект.

In [11]:

```
bs = pickle.dumps(dat1)
bs
```

Out[11]:

```
b'\x80\x03]q\x00(X\x0c\x00\x00\x00\xd0\xa1\xd1\x82\xd1\x80\xd0\xbe\xd0\xba\xd
0\xb0q\x01K\x0cK\x03\x86q\x02e.'
```

In [12]:

```
pickle.loads(bs)
```

Out[12]:

```
['Строка', (12, 3)]
```

Модуль shelve позволяет сохранять объекты под определенным ключом (задается в виде строки) и предоставляет интерфейс доступа, сходный со словарями. Для сериализации объекта используются возможности модуля `pickle` , а чтобы записать получивщуюся строку по ключу в файл, применяется модуль `pickle` . Все эти действия модуль shelve производит незаметно для нас.

Чтобы открыть файл с базой объектов, используется функция `open()` . Функция имеет следующий формат:

o pen (<Путь к файлу> [, flag="c"] [, protoco1=None] [, writeback=Fa1se])

В необязательном параметре flag можно указать один из режимов открытия файла:

- r - только чтение;
- w - чтение и запись;
- c - чтение и запись (значение по умолчанию). Если файл не существует, он будет создан;
- n - чтение и запись. Если файл не существует, он будет создан. Если файл существует, он будет перезаписан.

Функция `open()` возвращает объект, с помощью которого производится дальнейшая работа с базой данных. Этот объект имеет следующие методы:

- `close()` - закрывает файл с базой данных.
- `keys()` - возвращает объект с ключами;
- `values()` - возвращает объект to значениями;
- `items()` - возвращает объект, поддерживающий итерации. На каждой итерации возвращается кортеж, содержащий ключ и значение.
- `get(<Ключ> [, <Значение по умолчанию>] )` - если ключ присутствует; то метод возвращает значение, соответствующее этому ключу. Если ключ отсутствует, то возвращается начение None или значение, указанное во втором параметре;
- `setdefault(<Ключ> [, <Значение по умолчанию>] )` ---: если ключ nрисутствует, то метод возвращает значение, соответствующее этому ключу. Если ключ отсутствует, то вставляет новый элемент со значением, указанным во втором параметре, и возвращает это значение. Если второй параметр не указан, значением нового элемента будет None;
- `pop (<Ключ> [, <Значение по умолчанию>] )` - удаляет элемент с указанным ключом и возвращает его значение. Если ключ отсутствует, то возвращается значение из второго параметра. Если ключ отсутствует, и второй параметр не указан, то возбуждается исключение KeyError;

- `popitem()` - удаляет произвольный элемент и возвращает кортеж из ключа и значения. Если файл пустой, возбуждается исключение KeyError;
- `clear ()` - удаляет все элементы. Метод ничего не возвращает в качестве значения;
- `update ()` - добавляет элементы. Метод изменяет текущий объект и ничего не возвращает. Если элемент с указанным ключом уже присутствует, то его значение будет перезаписано.

Помимо этих методов можно воспользоваться функцией `len()` для получения количества элементов и оператором `del` для удаления определенного элемента, а также оператором `in` для проверки существования ключа.

In [13]:

```
import shelve
```

In [14]:

```
db = shelve.open("shl_1")
```

In [15]:

```
db["obj1"] = [1, 2, 3, 4, 5]
db["obj2"] = (6, 7, 8, 9, 10)
```

In [16]:

```
db["obj1"]
```

Out[16]:

```
[1, 2, 3, 4, 5]
```

In [17]:

```
db["obj2"]
```

Out[17]:

```
(6, 7, 8, 9, 10)
```

In [18]:

```
db.close()
```

In [19]:

```
db = shelve.open('shl_1')
```

In [20]:

```
db.keys()
```

Out[20]:

```
KeysView(<shelve.DbfilenameShelf object at 0x0000020FA2B9F348>)
```

In [21]:

```
list(db.keys())
```

Out[21]:

```
['obj1', 'obj2']
```

In [22]:

```
list(db.values())
```

Out[22]:

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

In [23]:

```
for k, v in db.items():
    print('key: {}, value: {}'.format(k, v))
```

```
key: obj1, value: [1, 2, 3, 4, 5]
key: obj2, value: (6, 7, 8, 9, 10)
```

Восстанавливаем объекты из файла:

In [36]:

```
db = shelve.open('shl_1')
```

In [37]:

```
db.keys()
```

Out[37]:

```
KeysView(<shelve.DbfilenameShelf object at 0x0000024613C1D308>)
```

In [38]:

```
list(db.keys())
```

Out[38]:

```
['obj1', 'obj2']
```

In [39]:

```
list(db.values())
```

Out[39]:

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

In [40]:

```
for k, v in db.items():
    print('key: {}, value: {}'.format(k, v))
```

```
key: obj1, value: [1, 2, 3, 4, 5]
key: obj2, value: (6, 7, 8, 9, 10)
```

Формат JSON

- к оглавлению

- **JSON** (JavaScript Object Notation) - текстовый формат обмена данными, основанный на JavaScript.

Причины популярности JSON:

- JSON **легко понимать** (легко читается людьми, простой и понятный синтаксис).
- JSON'ом **легко манипулировать** (програмно и вручную).
- JSON **легко генерировать**.
- Несмотря на происхождение от JavaScript, формат **считается независимым от языка программирования** и может использоваться практически с любым языком программирования.
- Поскольку формат JSON является подмножеством синтаксиса языка JavaScript, то позволяет быстро сериализовать/десериализовать данные между сервером и браузером из-за чего **получил очень широкое распространение в веб-разработке**.
- За счёт своей лаконичности по сравнению с XML формат JSON **может быть более подходящим для сериализации сложных структур данных**.

Чем JSON не является:

- Форматом «документов».
- Языком разметк.
- Языком программирования.

Минусы JSON:

- Нет конструкций для комментирования данных.
- Нет пространства имен.
- Нет валидации.
- Не расширяемый.

Синтаксис JSON

- Массив: `[значение1, значение2, значениеN]`
- Объект (можно интерпретировать как словарь в Python): `{имя1:значение1, имя2:значение2, имяN:значениеN}`
- Сложный объект: `{имя1:значение1, имя2: {имя2_1:значение2_1,имя2_2:значение2_2} }`

Объекты и массивы в JSON являются конструкциями, а литералы непосредственно данными, которые группируются этими конструкциями.

Список литералов JSON:

- строка
- число
- логическое значение
- значение `null`

Пример: `{имя1:"строка", имя2:13, имя3:true, имя4:false, имя5:null}`

Создание JSON объектов:

Импорт библиотек:

In [24]:

```
import json # библиотека для работы с JSON
# import pandas as pd
import requests # библиотека для выполнения HTTP запросов
```

In [25]:

```
my_ab = [] # создаем адресную книгу
```

In [26]:

```
addr1 = {}
addr1['name'] = 'Faina Lee'
addr1['email'] = 'faina@mail.ru'
addr1['birthday'] = '22.08.1994'
addr1['phones'] = [{'phone': '232-19-55'},
                   {'phone': '+7 (916) 232-19-55'}]

my_ab.append(addr1)
```

In [27]:

```
addr2 = {}
addr2['name'] = 'Robert Lee'
addr2['email'] = 'robert@mail.ru'
addr2['birthday'] = '22.08.1994'
addr2['phones'] = [{'phone': '111-19-55'},
                   {'phone': '+7 (916) 445-19-55'}]
my_ab.append(addr2)
```

In [28]:

```
my_ab
```

Out[28]:

```
[{'name': 'Faina Lee',
  'email': 'faina@mail.ru',
  'birthday': '22.08.1994',
  'phones': [{'phone': '232-19-55'}, {'phone': '+7 (916) 232-19-55'}]},
 {'name': 'Robert Lee',
  'email': 'robert@mail.ru',
  'birthday': '22.08.1994',
  'phones': [{'phone': '111-19-55'}, {'phone': '+7 (916) 445-19-55'}]}]
```

In [29]:

```
my_ab2 = [{
        'birthday': '22.08.1994',
        'email': 'faina@mail.ru',
        'id': 3,
        'name': 'Faina Lee',
        'phones': [{'phone': '232-19-55', 'phone_type': 'work'},
                   {'phone': '+7 (916) 199-93-79', 'phone_type': 'cell'},
                   {'phone': '+7 (912) 172-33-27', 'phone_type': 'home'}]
    },{
        'birthday': '02.11.1991',
        'email': 'robert@yandex.ru',
        'id': 4,
        'name': 'Robert Lee',
        'phones': [{'phone': '+7 (912) 672-13-71', 'phone_type': 'home'}]
    }]
```

In [28]:

```
my_ab2
```

Out[28]:

```
[{'birthday': '22.08.1994',
  'email': 'faina@mail.ru',
  'id': 3,
  'name': 'Faina Lee',
  'phones': [{'phone': '232-19-55', 'phone_type': 'work'},
   {'phone': '+7 (916) 199-93-79', 'phone_type': 'cell'},
   {'phone': '+7 (912) 172-33-27', 'phone_type': 'home'}]},
 {'birthday': '02.11.1991',
  'email': 'robert@yandex.ru',
  'id': 4,
  'name': 'Robert Lee',
  'phones': [{'phone': '+7 (912) 672-13-71', 'phone_type': 'home'}]}]
```

In [30]:

```
with open('addres-book.json', mode='w', encoding='utf-8') as f: # открываем файл на запись
    json.dump(my_ab, f, indent=2)
```

Как выглядит файл:

In [31]:

```
with open('addres-book.json', 'r', encoding='utf-8') as f:
    for l in f:
        print(l)
```


```
[

  {

    "name": "Faina Lee",

    "email": "faina@mail.ru",

    "birthday": "22.08.1994",

    "phones": [

      {

        "phone": "232-19-55"

      },

      {

        "phone": "+7 (916) 232-19-55"

      }

    ]

  },

  {

    "name": "Robert Lee",

    "email": "robert@mail.ru",

    "birthday": "22.08.1994",

    "phones": [

      {

        "phone": "111-19-55"

      },

      {

        "phone": "+7 (916) 445-19-55"

      }

    ]

  }
```

```
]
```

In [32]:

```
with open('addres-book.json', 'r', encoding='utf-8') as f: # открываем файл на чтение
    ab_fjs = json.load(f)
```

In [33]:

```
ab_fjs
```

Out[33]:

```
[{'name': 'Faina Lee',
  'email': 'faina@mail.ru',
  'birthday': '22.08.1994',
  'phones': [{'phone': '232-19-55'}, {'phone': '+7 (916) 232-19-55'}]},
 {'name': 'Robert Lee',
  'email': 'robert@mail.ru',
  'birthday': '22.08.1994',
  'phones': [{'phone': '111-19-55'}, {'phone': '+7 (916) 445-19-55'}]}]
```

In [34]:

```
ab_fjs[0]['name']
```

Out[34]:

```
'Faina Lee'
```

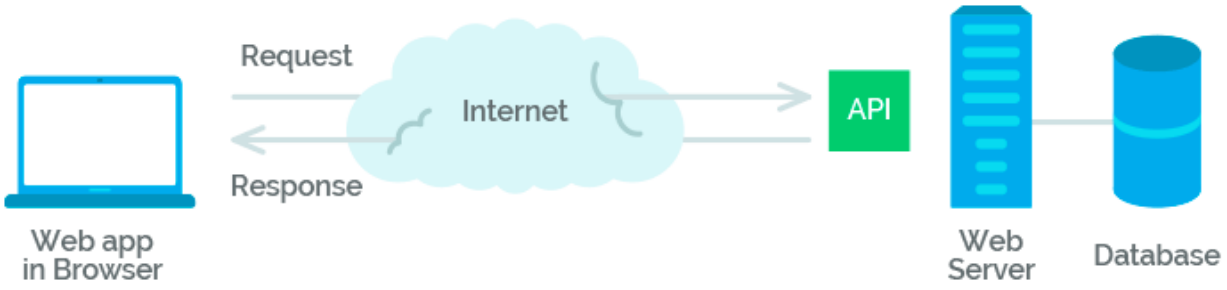
In [35]:

```
ab_fjs[1]['name']
```

Out[35]:

```
'Robert Lee'
```

Получение данных в формате JSON из публичных REST API



Работа REST API

REST (от англ. Representational State Transfer — «передача состояния представления») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети.

REST – это архитектурный стиль, а **RESTful API** – это его практическое воплощение.

RESTful API сводится к четырем базовым операциям:

- получение данных в удобном для клиента формате
- создание новых данных
- обновление данных
- удаление данных

REST функционирует поверх протокола HTTP, поэтому стоит упомянуть о его основных особенностях. Для каждой операции указанной выше используется свой собственный HTTP метод:

- **GET** – получение
- **POST** – создание
- **PUT** – обновление, модификация
- **DELETE** – удаление

Сопоставление методов HTTP запросам SQL и CRUD нотации

Публичное API для работы с почтовыми индексами: [http://api.zippopotam.us (http://api.zippopotam.us)](http://api.zippopotam.us)

In [36]:

```
ZIPPOPOTAM = 'http://api.zippopotam.us'
COUNTRY = 'RU'
zip_1 = '125009'
zip_2 = '129337'
```

In [37]:

```
'/'.join((ZIPPOPOTAM, COUNTRY, zip_1))
```

Out[37]:

```
'http://api.zippopotam.us/RU/125009'
```

In [38]:

```
def url_rus_zip(zip_code):
    return '/'.join((ZIPPOPOTAM, COUNTRY, zip_code))
```

In [39]:

```
url_rus_zip(zip_1)
```

Out[39]:

```
'http://api.zippopotam.us/RU/125009'
```

Краткая документация по библиотеке requests: http://docs.python-requests.org/en/master/user/quickstart/ (http://docs.python-requests.org/en/master/user/quickstart/)

In [40]:

```
# responce for HTTP GET request from http://api.zippopotam.us
r = requests.get(url_rus_zip(zip_2))
r
```

Out[40]:

```
<Response [200]>
```

In [41]:

```
r.content
```

Out[41]:

```
b'{"post code": "129337", "country": "Russia", "country abbreviation": "RU",
"places": [{"place name": "\\u041c\\u043e\\u0441\\u043a\\u0432\\u0430 337",
"longitude": "45.6667", "state": "\\u041c\\u043e\\u0441\\u043a\\u0432\\u043
0", "state abbreviation": "", "latitude": "60.15"}]}'
```

In [42]:

```
# Convert response into a python object
data = r.json()
```

In [43]:

```
# View the data
data
```

Out[43]:

```
{'post code': '129337',
 'country': 'Russia',
 'country abbreviation': 'RU',
 'places': [{'place name': 'Москва 337',
   'longitude': '45.6667',
   'state': 'Москва',
   'state abbreviation': '',
   'latitude': '60.15'}]}
```

Разбор данных

In [44]:

```
# One level deep
data['places']
```

Out[44]:

```
[{'place name': 'Москва 337',
  'longitude': '45.6667',
  'state': 'Москва',
  'state abbreviation': '',
  'latitude': '60.15'}]
```

In [45]:

```
# One level deep, second element
data['places'][0]
```

Out[45]:

```
{'place name': 'Москва 337',
 'longitude': '45.6667',
 'state': 'Москва',
 'state abbreviation': '',
 'latitude': '60.15'}
```

In [46]:

```
# One level down, then second item, then it's longitude object
data['places'][0]['longitude']
```

Out[46]:

```
'45.6667'
```

Цикл для вывода координат (широты и долготы) всех мест:

In [47]:

```
def extract_latlong(json):
    # For each element, i, in data.places,
    for i in data['places']:
        # print the latitude element and the longitude element
        print(i['latitude'], i['longitude'])
```

In [48]:

```
# Run the function
extract_latlong(data)
```

```
60.15 45.6667
```

Формат XML

- к оглавлению

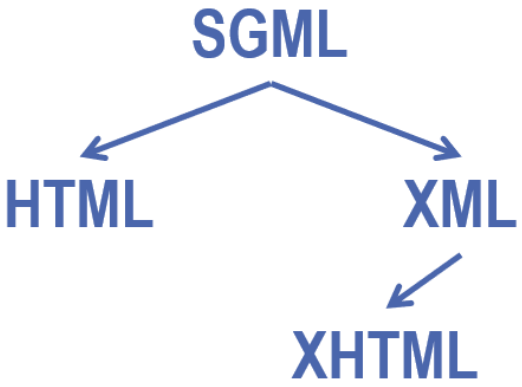
XML (англ. eXtensible Markup Language) — расширяемый язык разметки.

Цели создания:

- Эффективное описание (структурирование) данных.
- Обмен данными между информационными системами (в первую очередь – через интернет).

Основные особенности:

- Простой синтаксис.
- Удобство создания и обработки документов программами.
- Удобство чтения и создания документов человеком (профессиональными пользователями).
- Удобство обмена документами в интернете.
- Гибкость применения и расширяемость - возможнсть создавать собственные расширения XML.
- Очень широкая распространенность и доступность инструментов.
- **Расширение XML** — это конкретная грамматика, созданная на базе XML и представленная словарём тегов и их атрибутов, а также набором правил, определяющих какие атрибуты и элементы могут входить в состав других элементов.



Семейство языков SGML

История вопроса:

1. 1969 году в IBM разработан язык **GML** (Generalized Markup Language).
2. В 80-е на основе GML разрабатывается язык **SGML** (Standard Generalized Markup Language) — стандартный обобщённый язык разметки, метаязык, на котором можно определять язык разметки для документов. HTML и XML произошли от SGML.
3. **HTML** — (HyperText Markup Language — «язык гипертекстовой разметки») это стандартный язык разметки документов в WWW. Большинство веб-страниц содержат описание разметки на языке HTML. HTML это приложение SGML (создан в 1991 г.)
4. **XML** (англ. eXtensible Markup Language) подмножество SGML, разработанное для упрощения процесса машинного разбора документа.создан в 1997 г.
5. **XHTML** — расширяемый язык гипертекстовой разметки. XHTML, это семейство языков разметки веб-страниц на основе XML,

Сравнение XML и HTML:

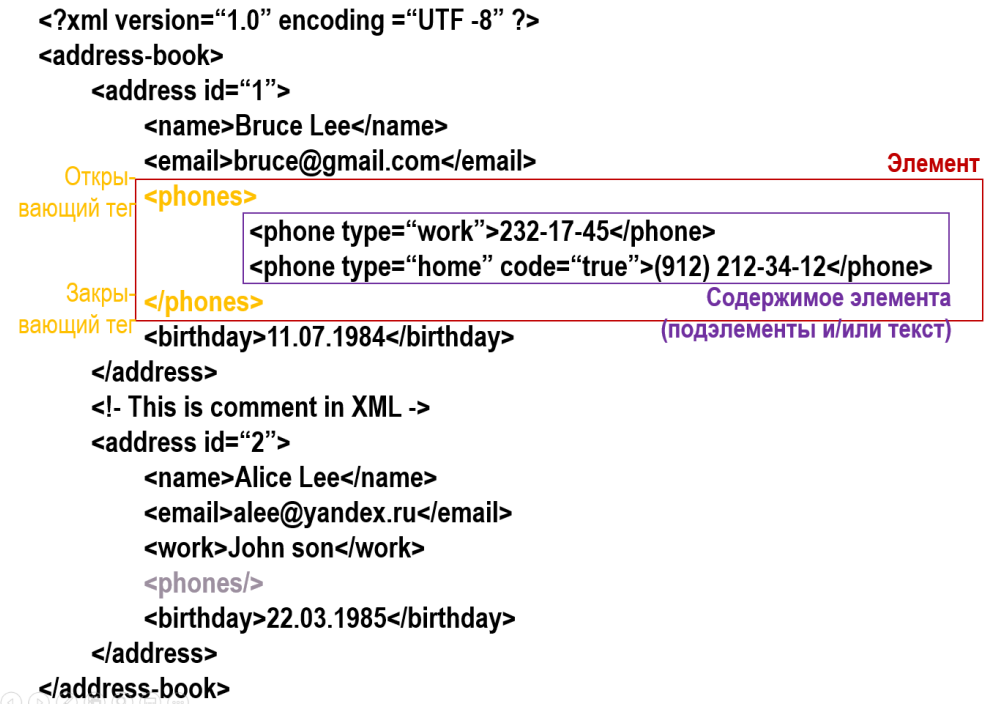
XML	HTML
Фиксированное множество тегов.	Расширяемое множество тегов.
Формат ориентирован на описание представления (внешнего вида) документа.	Формат ориентирован на содержимое документа.
Единственное представление данных.	Доступно множество форм представления документа.

XML	HTML
Нет возможностей валидации данных.	Есть возможность валидации данных. Высокие требования к корректности разметки данных.

Пример XML:

```
<?xml version=“1.0” encoding =“UTF-8” ?>
<address-book>
<address id=“1”>
    <name>Bruce Lee</name>
    <email>bruce@gmail.com</email>
    <phones>
        <phone type=“work”>232-17-45</phone>
        <phone type=“home” code=“true”>(912) 212-34-12</phone>
    </phones>
    <birthday>11.07.1984</birthday>
</address>
<!- This is comment in XML ->
<address id=“2”>
    <name>Alice Lee</name>
    <email>alee@yandex.ru</email>
    <work>John son</work>
    <phones/>
    <birthday>22.03.1985</birthday>
</address>
</address-book>
```

Элементы и теги



XML: элементы и теги

- **Теги используются парами**: открывающий тег (например: `<tag-name />`) – закрывающий тег (например: `</tag-name>`).
- **Элемент не имеющий содержимого** может описываться одним тегом вида: `<tag-name />`

- Для элементов должна выполнятся **вложенность**.
- У документа должен быть **единственный корневой элемент**.
- Имена тегов **чувствительны к регистру**.

Атрибуты

```
<?xml version="1.0" encoding ="UTF -8" ?>
<address-book>
    <address id="1">
        <name>Bruce Lee</name>
        <email>bruce@gmail.com</email>
        <phones>
            <phone type="work">232-17-45</phone>
            <phone type="home" code="true">(912) 212-34-12</phone>
        </phones>
        <birthday>11.07.1984</birthday>
    </address>
    <!- This is comment in XML ->
    <address id="2">
        <name>Alice Lee</name>
        <email>alee@yandex.ru</email>
        <work>John&amp;son</work>
        <phones/>
        <birthday>22.03.1985</birthday>
    </address>
</address-book>
```

Атрибуты (с именем и значением)

XML: атрибуты

- В элементе может быть только один атрибут с данным именем.
- Атрибуты **не имеют структуры** (только строка с содержимым).
- Значение атрибута должно заключаться в кавычки.
- Правило использования: **содержимое в элементах, метаданные – в атрибутах**.

Специальные символы

```
<?xml version="1.0" encoding ="UTF -8" ?>
<address-book>
    <address id="1">
        <name>Bruce Lee</name>
        <email>bruce@gmail.com</email>
        <phones>
            <phone type="work">232-17-45</phone>
            <phone type="home" code="true">(912) 212-34-12</phone>
        </phones>
        <birthday>11.07.1984</birthday>
    </address>
    <!- This is comment in XML ->
    <address id="2">
        <name>Alice Lee</name>
        <email>alee@yandex.ru</email>
        <work>John&amp;son</work>
        <phones/>
        <birthday>22.03.1985</birthday>
    </address>
</address-book>
```

Пролог

Комментарий

XML: специальные символы

- Некоторые специальные символы должны быть заэкранированы (escaped) при помощи сущностей (entities):

    - `<` → `<`
    - `&` → `&`
    - `>` → `>`
    - `“` → `&quot;`
    - `‘` → `&apos;`
- Тэги не могут содержать `<` или `&`

Корректность и действительность XML

Корректный XML документ

- **Корректный (well-formed) XML документ** соответствует всем общим правилам синтаксиса XML применимым к любому XML-документу. В частности:
    - правильная структура документа
    - совпадение имен в начальном и конечном теге элемента и т. п.
- Документ, который неправильно построен (т.е. не является корректным XML документом), не может считаться документом XML.

- Документ является **действительным XML документом** (valid), если с ним связано объявление типа документа и документ отвечает ограничениям, представленным в объявлении типа.
- Способами объявления типа являются:
    - Document type definition (DTD) - самый ранний способ определения типа.
    - XML Schema definition (XSD) - более современный вариант.
    - RELAX NG (Regular Language for XML Next Generation)
    - DSDL (Document Schema Definition Languages)
- **XML процессоры (parsers)** могут проверять или не проверять дйствительность XML документа. Проверяющие процессоры проверяют действительность документа и должны сообщать (по выбору пользователя) о нарушении ограничений, сформулированных в объявлении типа документа.
- Для большого количества прикладных областей созданы общедоступные объявления типов XML документов, что упрощает процедуру обмена данными между информационными системами.

Примеры объявления схемы XML документов

- Пример XML со ссылкой на DTD:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "http://www.w3schools.com/xml/note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- Пример XML со ссылкой на XML Schema (схема хранится в файле `note.xsd`):

```xml
<?xml version="1.0"?>
<note xmlns="http://www.w3schools.com" xmlns:xsi="http://www.w3.org/2001/XMLSch
ema-instance" xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- XML Schema:
  - Используется для тех же целей, что и схема БД.
  - Имеет набор предопределенных простых типов (строки, целые числа и т.п.)
  - Позволяет определять собственные сложные типы
  - Спецификация XML Schema является рекомендацией W3C.
- Пример XML Schema (схема хранится в файле `note.xsd`):

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://ww
w.w3schools.com" xmlns="http://www.w3schools.com" elementFormDefault="qualified">
    <xs:element name="note">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="to" type="xs:string"/>
                <xs:element name="from" type="xs:string"/>
                <xs:element name="heading" type="xs:string"/>
                <xs:element name="body" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Использование различных типов документов XML

- Для большого количества прикладных областей созданы общедоступные объявления типов XML документов, что упрощает процедуру обмена данными между информационными системами.

Примеры широко известных типов документов XML :

- XHTML — версия HTML, отвечающая синтаксическим требованиям XML.
- OpenDocument Format, Office Open XML — форматы файлов для офисных документов (Open Office и MS Office).
- FB2 — формат описания книг, базирующийся на XML.
- XBRL (eXtensible Business Reporting Language) — расширяемый язык деловой отчётности: широко используемый в мире открытый стандарт обмена деловой информацией. XBRL позволяет выражать с помощью семантических средств общие для участников рынка и регулирующих органов требования к представлению бизнес-отчётности.
- RSS (Rich Site Summary) — семейство XML-форматов, предназначенных для описания лент новостей, анонсов статей, изменений в блогах и т. п. Информация из различных источников, представленная в формате RSS, может быть собрана, обработана и представлена пользователю в удобном для него виде специальными программами-агрегаторами или онлайн-сервисами

Существует огромное количество форматов документов для различных предметных областей. Списки только некоторых типов документов XML:

- https://en.wikipedia.org/wiki/List_of_XML_markup_languages (https://en.wikipedia.org/wiki/List_of_XML_markup_languages)
- https://en.wikipedia.org/wiki/List_of_types_of_XML_schemas#Math_and_science (https://en.wikipedia.org/wiki/List_of_types_of_XML_schemas#Math_and_science)

JSON vs XML

```
{
    "sessionStart": "16-03-18-12-33-09",
    "sessionEnd": "16-03-18-12-33-12",
    "mapName": "TestMap",
    "logSections": [{
        "sector": {
            "x": 2.0,
            "y": -1.0,
            "z": 0.0
        },
        "logLines": [{
            "time": 37.84491729736328,
            "state": 0,
            "action": 1,
            "playerPosition": {
                "x": 24.560218811035158,
                "y": -8.940696716308594e-8,
                "z": 3.3498525619506838
            },
            "cameraRotation": {
                "x": 0.24549755454063416,
                "y": 0.017123013734817506,
                "z": 0.031348951160907748,
                "w": -0.9687389135360718
            },
    ...
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
    <sessionStart>16-03-18-12-33-09</sessionStart>
    <sessionEnd>16-03-18-12-33-12</sessionEnd>
    <mapName>TestMap</mapName>
    <logSections>
        <sector>
            <x>2</x>
            <y>-1</y>
            <z>0</z>
        </sector>
        <logLines>
            <time>37.84491729736328</time>
            <state>0</state>
            <action>1</action>
            <playerPosition>
                <x>24.560218811035156</x>
                <y>-8.940696716308594e-8</y>
                <z>3.3498525619506836</z>
            </playerPosition>
            <cameraRotation>
                <x>0.24549755454063416</x>
                <y>0.017123013734817505</y>
                <z>0.031348951160907745</z>
                <w>-0.9687389135360718</w>
            </cameraRotation>
    ...
```

JSON vs XML

JSON схож с XML по следующим пунктам:

- Текстовые форматы.
- Удобство чтения и создания документов человеком («само описывающие форматы»).
- Иерархические (значения могут содержать списки объектов или значений).

Когда использовать JSON а когда XML?

- JSON предпочтительнее в простых приложениях и позволяет удовлетворить простым требованиям по обмену данными.
- XML предпочтительнее для приложений со сложными требованиями к обмену данными, например, в корпоративном секторе.

JSON проще XML, но XML имеет более мощный инструментарий. Для обычных задач краткая семантика JSON позволяет получать более простой код. Для приложений со сложными требованиями к обмену данными, например, на предприятии, мощные функции XML могут значительно снизить риски.

JSON отличается от XML по следующим пунктам:

Преимущества JSON:

- **+** Легче и быстрее чем XML.
- **+** JSON использует типизированные объекты. Все значения XML являются строками и должны разбираться во время исполнения.
- **+** Меньше синтаксиса, совсем нет семантики.

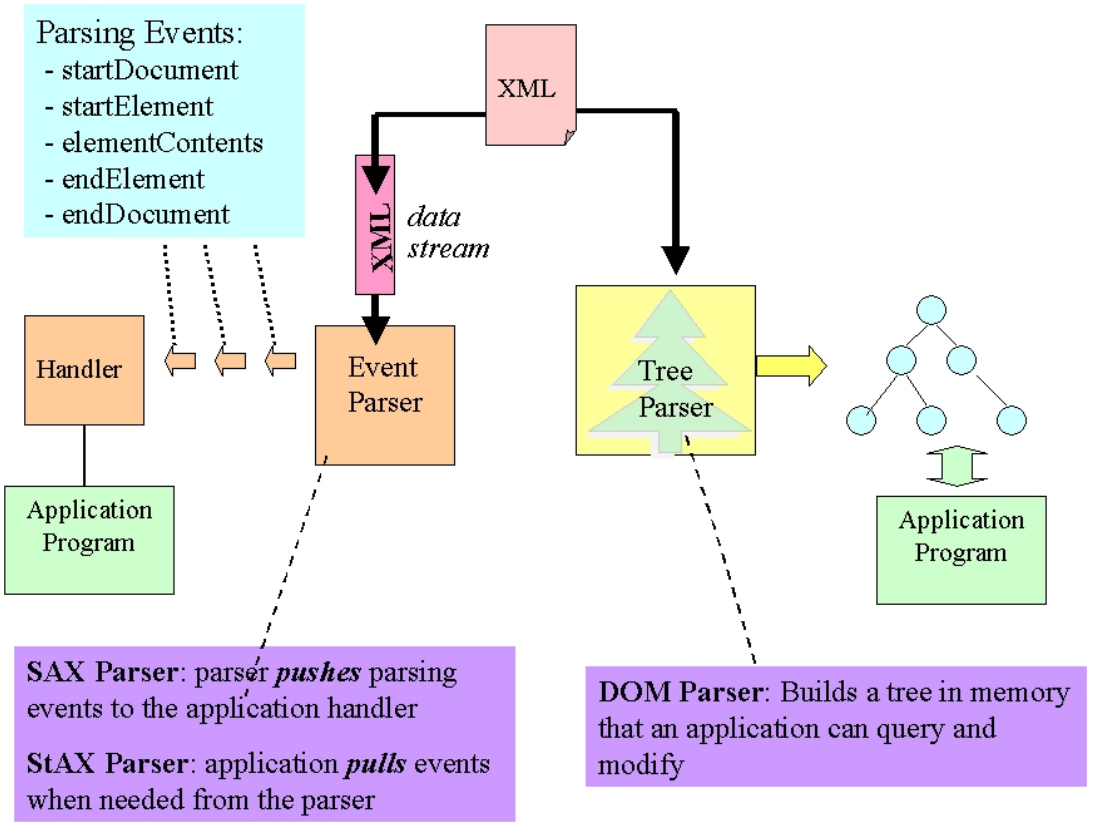
Преимущества XML:

- **+** Наличие пространств имен (namespace). Позволяет расширять язык документа за счет использования различных схем.

- **+** Атрибуты позволяют эффективно добавлять метаданные в документ. В JSON для этой цели приходится использовать ситуативные решения.
- **+** Поддержка действительности документа, позволяет автоматически проверять соответствие документа спецификации.
- **+** Множество дополнительных инструментов для работы с документами (XPath, XSL, XQuery), позволяющие обращаться к фрагментам документов и преобразоывать их.

JSON is best for simple applications, developed to satisfy simple requirements surrounding data interchange. XML is best for applications with complex requirements surrounding data interchange, such as in enterprise.

Работа с данными XML в приложениях



Сравнение SAX и DOM парсеров

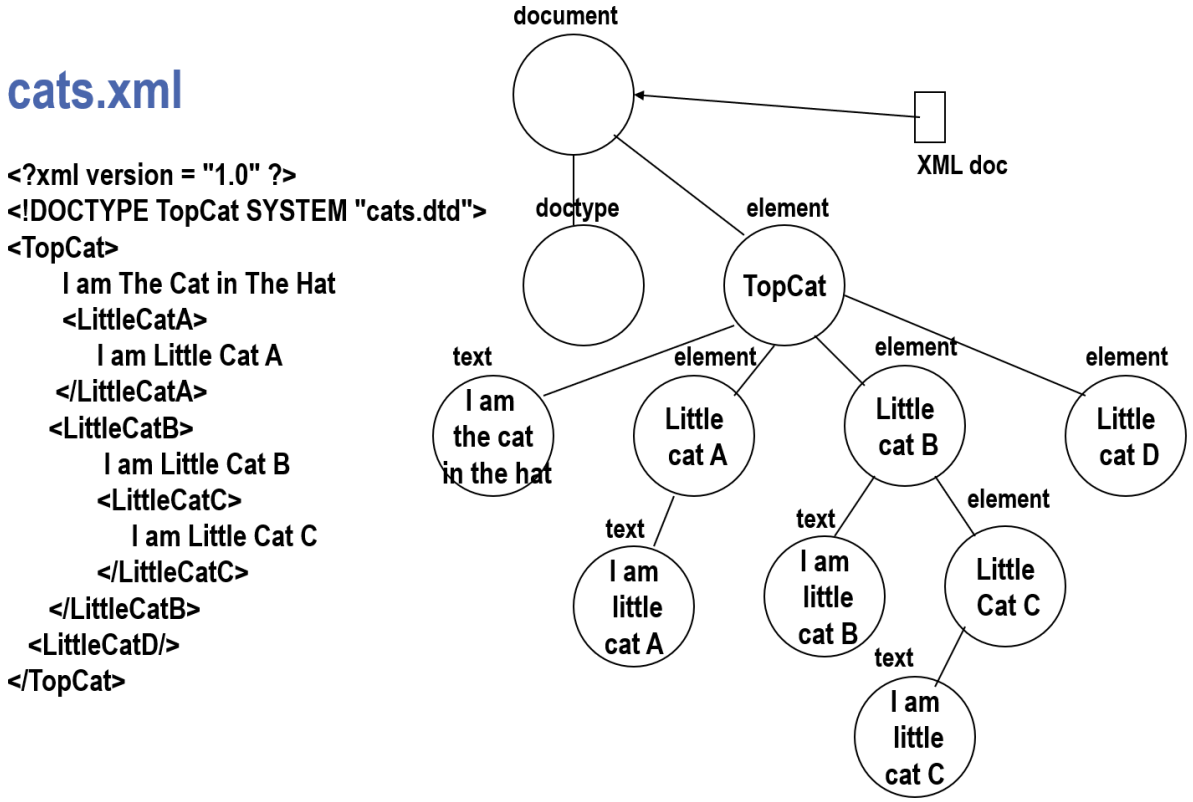
DOM (Document Object Model):

- **+** Естественное соответствие древовидной структуры документа и его объектной модели.
- **+** Возможность навигироваться по документу в любом направлении.
- **-** Необходимо прочитать весь документ в память.

SAX (Simple API for XML)

- Событийный подход к разбору XML.
- **-?** Более сложная логика работы с данными при их сложной организации.
- **-** Только последовательное получение информации из документа.
- **+** Можно обрабатывать только некоторую часть документа.
- **+** Позволяет не хранить весь документ в памяти.

DOM



Пример представления XML документа в виде DOM-дерева

Работа с XML

- к оглавлению

BeautifulSoup является библиотекой Python для парсинга HTML и XML документов. Часто используется для скрапинга веб-страниц. BeautifulSoup позволяет трансформировать XML или HTML документ в древо объектов Python, аналогичное DOM-дереву. Элементами этого дерева могут быть теги, навигация или комментарии.

Документация по BeautifulSoup:

- https://www.crummy.com/software/BeautifulSoup/ (https://www.crummy.com/software/BeautifulSoup/)
- https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (https://www.crummy.com/software/BeautifulSoup/bs4/doc/)
- Примеры работы BeautifulSoup с HTML: https://python-scripts.com/beautifulsoup-html-parsing (https://python-scripts.com/beautifulsoup-html-parsing)

Импорт библитек:

In [50]:

```
# import required modules
import requests
from bs4 import BeautifulSoup
```

Пример для работы:

```
<?xml version="1.0" encoding="UTF-8" ?>
<address_book>
<address id="1">
    <name>Bruce Lee</name>
    <email>bruce@gmail.com</email>
    <phones>
        <phone type="work">232-17-45</phone>
        <phone type="home" code="true">+7 (912) 212-34-12</phone>
    </phones>
    <birthday>11.07.1984</birthday>
</address>
<!-- This is comment in XML -->
<address id="2">
    <name>Alice Lee</name>
    <email>alee@yandex.ru</email>
    <work>John&amp;amp;son</work>
    <phones/>
    <birthday>22.03.1985</birthday>
</address>
</address_book>
```

In [51]:

```
# Выполняем парсинг XML файла:
with open('.\\addres-book.xml') as f:
    ab = BeautifulSoup(f, 'xml')
```

In [53]:

```
ab
```

Out[53]:

```
bs4.BeautifulSoup
```

In [79]:

```
# Переходим к корню дерева документа:
ab.address_book
```

Out[79]:

```
<address_book>
<address id="1">
<name>Bruce Lee</name>
<email>bruce@gmail.com</email>
<phones>
<phone type="work">232-17-45</phone>
<phone code="true" type="home">+7 (912) 212-34-12</phone>
</phones>
<birthday>11.07.1984</birthday>
</address>
<!-- This is comment in XML -->
<address id="2">
<name>Alice Lee</name>
<email>alee@yandex.ru</email>
<work>John&amp;son</work>
<phones/>
<birthday>22.03.1985</birthday>
</address>
</address_book>
```

In [53]:

```
ab.address_book.address
```

Out[53]:

```
<address id="1">
<name>Bruce Lee</name>
<email>bruce@gmail.com</email>
<phones>
<phone type="work">232-17-45</phone>
<phone code="true" type="home">+7 (912) 212-34-12</phone>
</phones>
<birthday>11.07.1984</birthday>
</address>
```

In [54]:

```
ab.address_book.address.name
```

Out[54]:

```
'address'
```

In [55]:

```
ab.address_book.address['id']
```

Out[55]:

```
'1'
```

In [56]:

```
ab.address_book.address.text
```

Out[56]:

```
'\nBruce Lee\nbruce@gmail.com\n\n232-17-45\n+7 (912) 212-34-12\n\n11.07.1984
\n'
```

In [57]:

```
ab.address_book.address.getText('|',strip=True) # Получаем все дочерние строки разделенные
```

Out[57]:

```
'Bruce Lee|bruce@gmail.com|232-17-45|+7 (912) 212-34-12|11.07.1984'
```

In [58]:

```
ab.address_book.address.contents
```

Out[58]:

```
['\n',
 <name>Bruce Lee</name>,
 '\n',
 <email>bruce@gmail.com</email>,
 '\n',
 <phones>
 <phone type="work">232-17-45</phone>
 <phone code="true" type="home">+7 (912) 212-34-12</phone>
 </phones>,
 '\n',
 <birthday>11.07.1984</birthday>,
 '\n']
```

In [59]:

```
# обходим все дочерние элементы address:
for ch in ab.address_book.address.children:
    print(ch.name,'->', repr(ch))
```

```
None -> '\n'
name -> <name>Bruce Lee</name>
None -> '\n'
email -> <email>bruce@gmail.com</email>
None -> '\n'
phones -> <phones>
<phone type="work">232-17-45</phone>
<phone code="true" type="home">+7 (912) 212-34-12</phone>
</phones>
None -> '\n'
birthday -> <birthday>11.07.1984</birthday>
None -> '\n'
```

In [60]:

```
# получаем первый элемент соответствующий указанному пути:
ab.address_book.address.phone
```

Out[60]:

```
<phone type="work">232-17-45</phone>
```

In [61]:

```
# получаем все дочерние элементы 'phone' для пути address_book.address:
ab.address_book.address.find_all('phone')
```

Out[61]:

```
[<phone type="work">232-17-45</phone>,
 <phone code="true" type="home">+7 (912) 212-34-12</phone>]
```

In [62]:

```
# ищем элементы соответствующие более сложному условию:
ab.address_book.address.find('phone', type='home')
```

Out[62]:

```
<phone code="true" type="home">+7 (912) 212-34-12</phone>
```

In [63]:

```
ph1 = ab.address_book.address.find('phone')
```

In [64]:

```
ph1
```

Out[64]:

```
<phone type="work">232-17-45</phone>
```

In [65]:

```
list(ph1.next_siblings)
```

Out[65]:

```
['\n', <phone code="true" type="home">+7 (912) 212-34-12</phone>, '\n']
```

In [66]:

```
ph1.next
```

Out[66]:

```
'232-17-45'
```

In [101]:

```
list(ph1.nextGenerator())
```

Out[101]:

```
['232-17-45',
 '\n',
 <phone code="true" type="home">+7 (912) 212-34-12</phone>,
 '+7 (912) 212-34-12',
 '\n',
 '\n',
 <birthday>11.07.1984</birthday>,
 '11.07.1984',
 '\n',
 '\n',
 ' This is comment in XML ',
 '\n',
 <address id="2">
 <name>Alice Lee</name>
 <email>alee@yandex.ru</email>
 <work>John&amp;son</work>
 <phones/>
 <birthday>22.03.1985</birthday>
 </address>,
 '\n',
 <name>Alice Lee</name>,
 'Alice Lee',
 '\n',
 <email>alee@yandex.ru</email>,
 'alee@yandex.ru',
 '\n',
 <work>John&amp;son</work>,
 'John&son',
 '\n',
 <phones/>,
 '\n',
 <birthday>22.03.1985</birthday>,
 '22.03.1985',
 '\n',
 '\n']
```

In [102]:

```
ph1.findNextSibling()
```

Out[102]:

```
<phone code="true" type="home">+7 (912) 212-34-12</phone>
```

Более крупный пример:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<address_book>
<country name="algeria">
<address id="1">
    <gender>m</gender>
    <name>Aicha Barki</name>
    <email>aiqraa.asso@caramail.com</email>
    <position>Presidente</position>
    <company>Association Algerienne d'Alphabetisation Iqraa</company>
    <phones>
        <phone type="work">+ (213) 6150 4015</phone>
        <phone type="personal">+ (213) 2173 5247</phone>
    </phones>
</address>
</country>
<country name="angola">
<address id="2">
    <gender>m</gender>
    <name>Francisco Domingos</name>
    <email>frandomingos@hotmail.com</email>
    <position>Directeur General</position>
    <company>Institut National de Education des Adultes</company>
    <phones>
        <phone type="work">+ (244-2) 325 023</phone>
        <phone type="personal">+ (244-2) 325 023</phone>
    </phones>
</address>
<address id="3">
    <gender>f</gender>
    <name>Maria Luisa</name>
    <email>luisagrilo@ebonet.net</email>
    <position>Directrice Nationale</position>
    <company>Institut National de Education des Adultes</company>
    <phones>
        <phone type="personal">+ (244) 4232 2836</phone>
    </phones>
</address>
<address id="4">
    <gender>m</gender>
    <name>Abraao Chanda</name>
    <email>ineda@snet.co.ao</email>
    <position>Chef</position>
    <company>Institut National de Education des Adultes</company>
    <phones>
        <phone type="work">+ (244-2) 325 023</phone>
        <phone type="personal">+ (244-2) 325 023</phone>
    </phones>
</address>
</country>
<country name="argentina">
<address id="5">
    <gender>m</gender>
```

```
        <name>Beatriz Busaniche</name>
        <email>busaniche@caminandoutopias.org.ar</email>
        <position>Executive Director</position>
        <company>Universidad de Buenos Aires</company>
        <phones>
            <phone type="work">+ (54-11) 4784 1159</phone>
        </phones>
    </address>
    </country>
    <country name="australia">
    <address id="6">
        <gender>f</gender>
        <name>Francesca Beddie</name>
        <email>f.beddie@ala.asn.au</email>
        <position>Executive Director</position>
        <company>Adult Learning Australia</company>
        <phones>
            <phone type="work">+ (61-2) 6274 9500</phone>
            <phone type="personal">+ (61-2) 6274 9513</phone>
        </phones>
    </address>
    <address id="7">
        <gender>m</gender>
        <name>Graham John Smith</name>
        <email>grasm@connexus.net.au</email>
        <position>Secretary</position>
        <company>Disability Australia Ltd</company>
        <phones>
            <phone type="work">+ (61-3) 9807 4702</phone>
        </phones>
    </address>
    </country>

    </address_book>
```

In [106]:

```
with open('.\\addres-book-q.xml') as f:
    ab = BeautifulSoup(f, 'xml')
```

In [107]:

```
res = list();
for person in ab.address_book.find_all("address"):
    ph = [phones.next for phones in person.phones.find_all("phone")]
    res.append({person.find("name").next: ph})
res
```

Out[107]:

```
[{'Aicha Barki': ['+ (213) 6150 4015', '+ (213) 2173 5247']},
 {'Francisco Domingos': ['+ (244-2) 325 023', '+ (244-2) 325 023']},
 {'Maria Luisa': ['+ (244) 4232 2836']},
 {'Abraao Chanda': ['+ (244-2) 325 023', '+ (244-2) 325 023']},
 {'Beatriz Busaniche': ['+ (54-11) 4784 1159']},
 {'Francesca Beddie': ['+ (61-2) 6274 9500', '+ (61-2) 6274 9513']},
 {'Graham John Smith': ['+ (61-3) 9807 4702']}]
```

In [108]:

```
ph1
```

Out[108]:

```
<phone type="work">232-17-45</phone>
```

TODO: кодировки, requests (краулинг), mime type

CSV, numpy-native, parket, xlsx, sqlite