# Лекция 3: Работа с файлами данных

**Автор: Сергей Вячеславович Макрушин, 2022 г.**

e-mail: s-makrushin@yandex.ru (mailto:s-makrushin@yandex.ru)

V 0.5 22.09.2022

## Разделы:



-

In [161]:
```
# загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v2.css")
HTML(html.read().decode('utf-8'))
```

Out[161]:

# Файлы

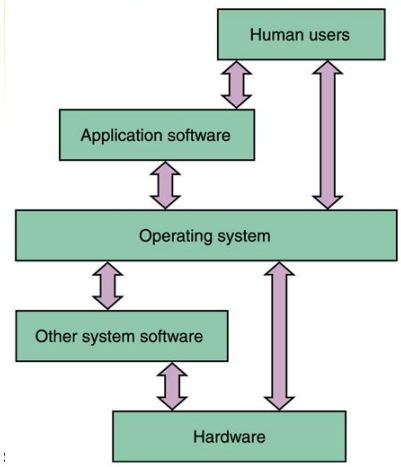
- к оглавлению

## Работа с файловой системой

- к оглавлению

#### Файл

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



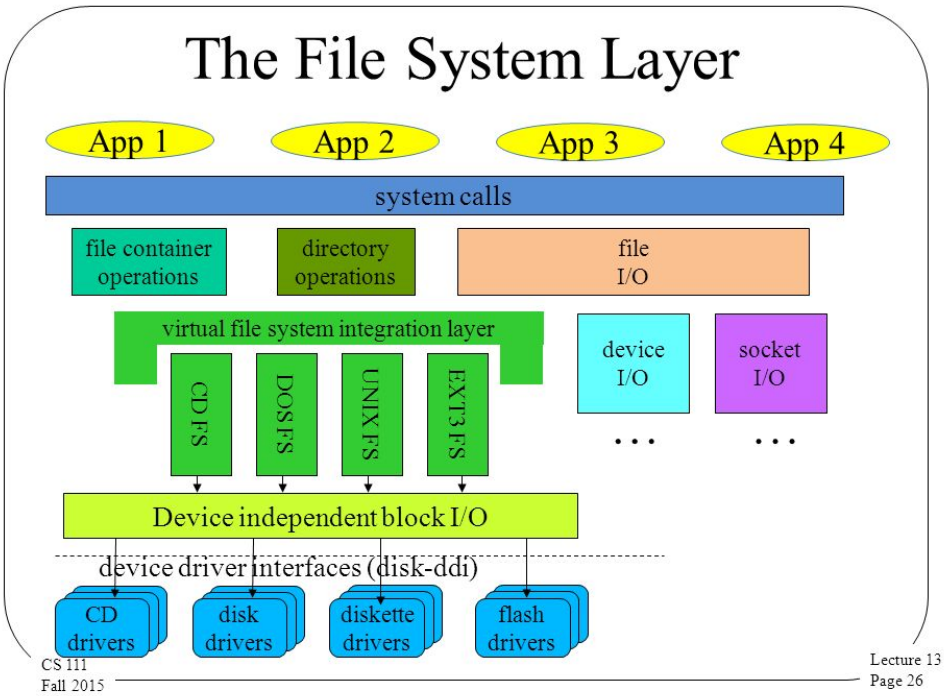

**Роль операционной системы**

- 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.

Про базовые операции работы с файлами см. Лекцию 5 курса Алгоритмы и структуры данных в языке программирования Python, часть 2: "Работа с файлами".

**Файловая система**

- **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.




**Уровни взаимодействия с файлами**

Базовые работы с файловой системой в Python:

In [162]:
```
ls
```

```
 ’®¬ ў гбва®©бвўЃ Е Ё¬ЃЃв ¬ЃвЄг Data
 ‘ЃаЁ©л© ®¬Ґа в®¬ : EE2C-D1DD

 ‘®¤Ґа¦Ё¬®Ґ Ї ЇЄЁ E:\YandexDisk\Python\Ipynb\TOBD_2022\03_data_files\lec

21.09.2022  06:52    <DIR>          .
21.09.2022  05:40    <DIR>          ..
21.09.2022  06:44    <DIR>          .ipynb_checkpoints
28.09.2021  15:46               463 addres-book.json
28.09.2021  15:46               517 addres-book.xml
28.09.2021  15:46             2я471 addres-book-q.xml
19.09.2022  15:53               762 changes.txt
28.09.2021  15:46                34 dat1.pickle
28.09.2021  15:46                51 dat2.pickle
21.09.2022  06:23    <DIR>          img
15.03.2021  09:29             2я835 lec_v2.css
21.09.2022  06:34                47 lec03_example.txt
28.09.2021  15:46               529 shl_1.dat
16.12.2021  15:15           117я836 TOBD_lec_03_data_files_v2.ipynb
22.03.2022  10:04           119я011 TOBD_lec_03_data_files_v2-Copy1 (2).ipynb
21.09.2022  06:52           197я418 TOBD_lec_03_data_files_v4.ipynb
21.09.2022  06:42             1я800 Untitled.ipynb
19.09.2022  12:11            50я908 ђ Ў®в  ў ®Ў« ЄҐ.ipynb
21.09.2022  06:48            24я915 ‘ЇҐ¦бЁ¬ў®«л_.xlsx
              15 д ©«®ў         519я597 Ў ©в
               4 Ї Ї®Є  84я095я660я032 Ў ©в бў®Ў®¤®
```

In [163]:
```
# базовый модуль для работы с функциями опреационной системы, в т.ч. работы с файловой с
import os
# модуль реализующий функцииии работы с путями в файловой системе:
import os.path
```

In [164]:
```
# получние текущей рабочей директории:
cwd = os.getcwd()
cwd
```

Out[164]:
```
'E:\\YandexDisk\\Python\\Ipynb\\TOBD_2022\\03_data_files\\lec'
```

In [167]:
```
# изменение текущей директории:
os.chdir('..') # поднимаемся на одну директорию вверх
cwd2 = os.getcwd()
cwd2
```

Out[167]:
```
'E:\\YandexDisk\\Python\\Ipynb\\TOBD_2022'
```

In [168]:
```
# типичная задача, получение списка директорий по опредленному пути:
from os import listdir
from os.path import isfile, join

# listdir - (ls) получение содержимого директории
# isfile -  проверка, является ли объект файлом или директорией
# join - корректный (инвариантный относительно ОС) способ конкатенации пути к файлу и им
onlyfiles = [f for f in listdir(cwd2) if isfile(join(cwd2, f))]
onlyfiles
```

Out[168]:
```
['addres-book.json',
 'L1_python_st.png',
 'pwd.txt',
 'tmp2.txt',
 'tobd_im.jpg',
 'TOBD_yt_.png',
 '~$Темы ТОБД_в2.xlsx',
 'Визуал_v2.pptx',
 'Нагрузка галактика 30.08.xlsx',
 'Нагрузка галактика 30.08_в2.xlsx',
 'ППС и Штатное расписание 2021-2022_ВЕРСИЯ август.xlsx',
 'РПД_ТОБД_ПИ_2021_в6.docx',
 'РПД_ТОБД_ПМ_2021_в6.docx',
 'Темы ТОБД.xlsx',
 'Темы ТОБД_в2.xlsx']
```

In [169]:
```
# рекурсивный обход дочерних директорий и файлов в них:
for root, dirs, files in os.walk(cwd2):
    print(f"{root}, dirs: {dirs}, files: {files}")
```

```
E:\YandexDisk\Python\Ipynb\TOBD_2022, dirs: ['01_numpy', '02_pandas', '03_data_file
s', 'TOBD22_for_students', 'TOBD22_for_teachers', 'TOBD22_MSV_sem', 'РПД'], files:
['addres-book.json', 'L1_python_st.png', 'pwd.txt', 'tmp2.txt', 'tobd_im.jpg', 'TOB
D_yt_.png', '~$Темы ТОБД_в2.xlsx', 'Визуал_v2.pptx', 'Нагрузка галактика 30.08.xls
x', 'Нагрузка галактика 30.08_в2.xlsx', 'ППС и Штатное расписание 2021-2022_ВЕРСИЯ
август.xlsx', 'РПД_ТОБД_ПИ_2021_в6.docx', 'РПД_ТОБД_ПМ_2021_в6.docx', 'Темы ТОБД.xl
sx', 'Темы ТОБД_в2.xlsx']
E:\YandexDisk\Python\Ipynb\TOBD_2022\01_numpy, dirs: ['lec', 'sem', 'tests', 'vide
o'], files: []
E:\YandexDisk\Python\Ipynb\TOBD_2022\01_numpy\lec, dirs: ['.ipynb_checkpoints', 'im
g'], files: ['ab_ndarr.npz', 'ab_ndarr.zip', 'a_ndarr.npy', 'b_s1.txt', 'b_s2.txt',
'lec_v1.css', 'lec_v2.css', 'TOBD20_lec_01_numpy_v8.pdf', 'TOBD20_lec_01_numpy_v9 -
Jupyter Notebook_.pdf', 'TOBD20_lec_01_numpy_v9.ipynb', 'TOBD20_lec_01_numpy_v9.pd
f', 'xx_ndarr.npz', 'Технический_v1.ipynb']
E:\YandexDisk\Python\Ipynb\TOBD_2022\01_numpy\lec\.ipynb_checkpoints, dirs: [], fil
es: ['TOBD20_lec_01_numpy_v9-checkpoint.ipynb', 'Технический_v1-checkpoint.ipynb']
E:\YandexDisk\Python\Ipynb\TOBD_2022\01_numpy\lec\img, dirs: [], files: ['L1_array_
structure.png', 'L1_axis0.png', 'L1_axis1.png', 'L1_axis2.png', 'L1_broadcasting.pn
g', 'L1_broadcasting1_.png', 'L1_broadcasting2.png', 'L1_broadcasting2_.png', 'L1_b
```

Полезные функции для работы с файловой системой:

- `os.getcwd()` - получение текущего пути
- `os.chdir()` - изменение текущего пути
- `os.mkdir()` - создание новой директории
- `os.rename()` - переименование директории
- `os.rmdir()` - удаление директории
- `os.walk()` - получение содержимого директории

---

# Передача файлов по сети

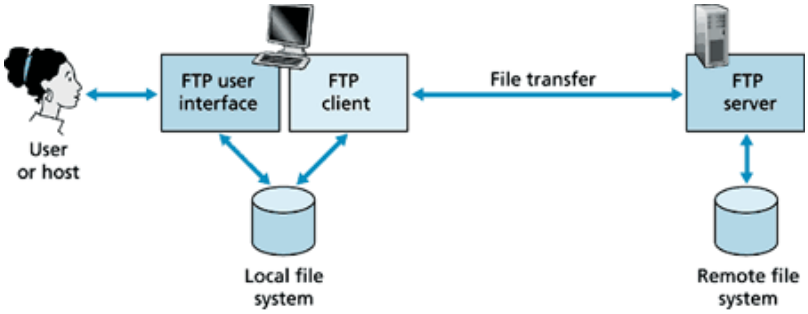

- к оглавлению


### Передача файлов по сети

- Имеются протоколы передачи данных, которые позволяют передавать файлы по сети, в частности: FTP, HTTP, WebDav.

### FTP

**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.




**Принцип использования протокола ftp**

**WebDav**

**HTTP (Hyper Text Transfer Protocol)** - это протокол передачи гипертекста (HTML - файлов). Протокол HTTP используется при пересылке Web-страниц между компьютерами, подключенными к одной сети.

**WebDav (Web Distribute Authoring and Resirping)** - набор расширений и дополнений к протоколу HTTP, поддерживающих совместную работу пользователей над редактированием файлов и управление файлами на удаленных веб-серверах.

- Изначально целью создания DAV являлась: "разработка дополнений к протоколу HTTP, обеспечивающих свободное взаимодействие инструментов распределённой разработки веб-страниц, в соответствии с потребностями работы пользователей".
- Однако на практики кроме коллективной работы над веб-документами WebDAV стал применяться в качестве **сетевой файловой системы, эффективной для работы в Интернете**.
  - способной обрабатывать файлы целиком
  - поддерживающей хорошую производительность в условиях окружения с высокой временной задержкой передачи информации

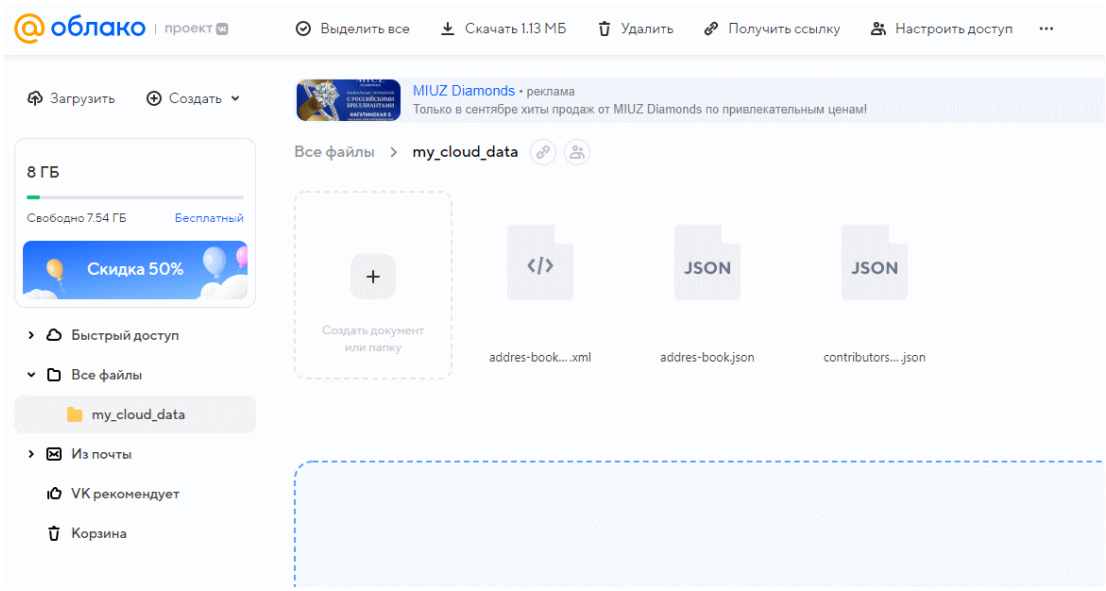
WebDAV широко применяется в качестве :

- протокола для доступа через Интернет и манипулирования содержимым систем документооборота. Ещё одной важной целью WebDAV
- инструмента поддержки работы распределённых команд по разработке программного обеспечения и обработки данных.

WebDAV расширяет средствами записи информации возможности HTTP в качестве стандартного уровня доступа к широкому кругу хранилищ информации.

**Работа с облачным хранилищем через браузер**

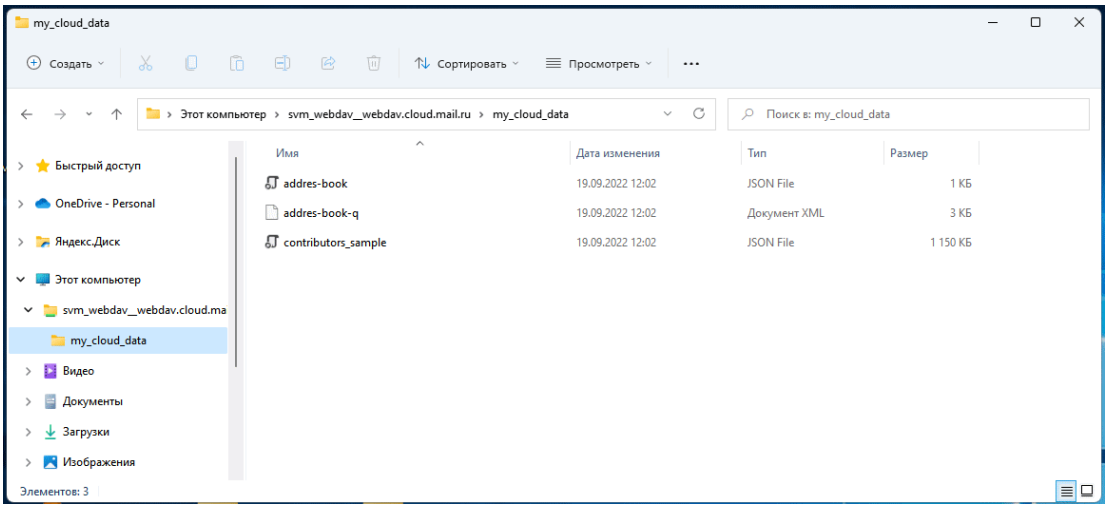
Классический пример работы с облачным файловым хранлищем через браузер (на примере Облака mail.ru):




**Интерфейс работы с облачным хранилищем в браузере**

### Подключение к облачному хранилищу через WebDav клиент

Подключение к фалам, хранящимся в облачном хранилище, через протокол webdav на примере работы с сервисом Облако mail.ru ( https://cloud.mail.ru (https://cloud.mail.ru) ). Инструкция по подключению к облаку по протоколу WebDav: https://help.mail.ru/cloud_web/app/webdav (https://help.mail.ru/cloud_web/app/webdav) (адрес для подключения сервиса: https://webdav.cloud.mail.ru (https://webdav.cloud.mail.ru)).




**Работа с облачным хранилищем через клиент WebDav (встроенный в MS Windows)**

In [29]:
```
# Установка библиотеки доступа по WebDav в Python:
!pip install webdavclient3
```

```
Collecting webdavclient3
  Downloading webdavclient3-3.14.6.tar.gz (23 kB)
Requirement already satisfied: requests in c:\users\alpha\.conda\envs\pytorch_1_5v2\li
b\site-packages (from webdavclient3) (2.22.0)
Requirement already satisfied: lxml in c:\users\alpha\.conda\envs\pytorch_1_5v2\lib\si
te-packages (from webdavclient3) (4.5.0)
Requirement already satisfied: python-dateutil in c:\users\alpha\.conda\envs\pytorch_1
_5v2\lib\site-packages (from webdavclient3) (2.8.1)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\users\alp
ha\.conda\envs\pytorch_1_5v2\lib\site-packages (from requests->webdavclient3) (1.25.8)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\users\alpha\.conda\envs\pyt
orch_1_5v2\lib\site-packages (from requests->webdavclient3) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\alpha\.conda\envs\pytorc
h_1_5v2\lib\site-packages (from requests->webdavclient3) (2019.11.28)
Requirement already satisfied: idna<2.9,>=2.5 in c:\users\alpha\.conda\envs\pytorch_1_
5v2\lib\site-packages (from requests->webdavclient3) (2.8)
Requirement already satisfied: six>=1.5 in c:\users\alpha\.conda\envs\pytorch_1_5v2\li
b\site-packages (from python-dateutil->webdavclient3) (1.14.0)
Building wheels for collected packages: webdavclient3
  Building wheel for webdavclient3 (setup.py): started
  Building wheel for webdavclient3 (setup.py): finished with status 'done'
  Created wheel for webdavclient3: filename=webdavclient3-3.14.6-py3-none-any.whl size
=20888 sha256=9bf35b5a377912415a85cc8c9e8c86d329c93880b3e8d77978f8716a7380defb
  Stored in directory: c:\users\alpha\appdata\local\pip\cache\wheels\ca\ab\bd\90226b06
142d06812d0dfea66a04ad2f684490d03f3fb139a5
Successfully built webdavclient3
Installing collected packages: webdavclient3
Successfully installed webdavclient3-3.14.6
```

In [170]:
```
# импорт:
from webdav3.client import Client
```

In [171]:
```
# настраиваем параметры подключения:
options = {
 'webdav_hostname': "https://webdav.cloud.mail.ru",
 'webdav_login':    "svm_webdav@mail.ru",
 'webdav_password': "тут пароль"
}
```

In [172]:
```
cd E:\YandexDisk\Python\Ipynb\TOBD_2022\03_data_files\
```

```
E:\YandexDisk\Python\Ipynb\TOBD_2022\03_data_files
```

In [176]:
```
# Из соображений конфиденциальности читаем пароль из файла:
with open("pwd.txt", "r") as f:
    for line in f:
        pwd = line

options['webdav_password'] = pwd
```

In [177]:
```
client = Client(options)
```

In [178]:
```
# Получить содержимое папки
client.list('/my_cloud_data/')
```

Out[178]:
```
['addres-book-q.xml', 'addres-book.json', 'contributors_sample.json']
```

In [179]:
```
# Скопировать папку удаленно в облаке (без скачивания на локальный компьютер):
client.copy(remote_path_from="/my_cloud_data", remote_path_to='/my_cloud_data2')
```

In [180]:
```
client.list('/my_cloud_data2/')
```

Out[180]:
```
['addres-book-q.xml', 'addres-book.json', 'contributors_sample.json']
```

In [181]:
```
# Скачать файл на локальный компьютер:
client.download(remote_path='/my_cloud_data/addres-book.json', local_path='addres-book.j
```

In [182]:
```
ls
```

```
 ’®¬ ў гбва®©бвўЃ Е Ё¬ЃЃв ¬ЃвЄг Data
 ‘ЃаЁ©л© ®¬Ґа в®¬ : EE2C-D1DD

 ‘®¤Ґа¦Ё¬®Ґ Ї ЇЄЁ E:\YandexDisk\Python\Ipynb\TOBD_2022\03_data_files

21.09.2022  07:35    <DIR>          .
19.09.2022  13:19    <DIR>          ..
21.09.2022  07:36               463 addres-book.json
21.09.2022  07:34    <DIR>          lec
19.09.2022  13:13                20 pwd.txt
16.09.2022  11:20    <DIR>          sem
               2 д ©«®ў            483 Ў ©в
               4 Ї Ї®Є  83я978я788я864 Ў ©в бў®Ў®¤®
```

In [183]:
```
# Загрузить файл с локального компьютера в облако
client.upload(remote_path='/addres-book2.json', local_path='addres-book.json')
```

In [184]:
```
client.list('/')
```

Out[184]:
```
['addres-book2.json', 'my_cloud_data/', 'my_cloud_data2/']
```

# Типы файлов


- к оглавлению


### 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.

**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 файлов**

**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.

**Текстовые и бинарные файлы**

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 "самозадокументрованные", 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

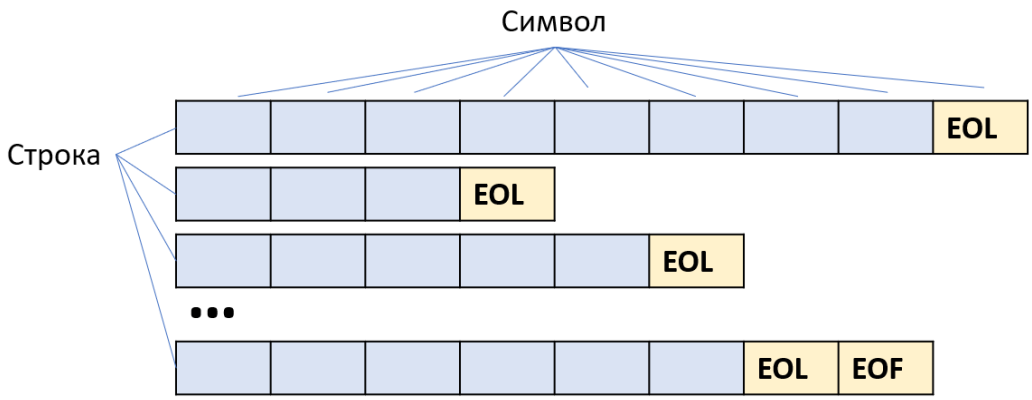
файла не выполняя никаких преобразований, что существенно ускоряет процесс ввода/вывода.
    - **+** Обычно хранение информации (в частности числовых массивов) в двоичном формате намного компактнее, что уменьшает требования к объему хранилища и увеличивает скорость ввода/вывода.
    - **±** Интерпретировать 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_%
    .

# Работа с текстовыми файлами

- к оглавлению

### Текстовые файлы

- **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.




**Логическая структура текстового файла**

Физически текстовый файл, как и двоичный, состоит из последовательности байт. Каждый символ кодируется одним или несколькими байтами. Символы конца строки или конца файла зависят от используемой операционной системы. Способ кодировки символов является метаинформацией файла.

### Кодировки (наборы символов)

- **Набор символов (character set)** — таблица, задающая кодировку множества символов алфавита (обычно элементов текста: букв, цифр, знаков препинания, спецсимволов). Такая таблица сопоставляет каждому символу последовательность длиной в один или несколько символов другого алфавита, в компьюетре, обычно в один или нескольким байтов.

    - Набор символов, который может использоваться несколькими языками. Пример: набор латинских символов используется в английском и большинстве европейских языков, хотя набор греческих символов используется только в греческом языке.
    - Хотя термин "набор символов" (character set, charset) сейчас является наиболее авторитетным, предшествовавший ему термин **"кодировка" (encoding)** по-прежнему **используется в качестве синонима**.
    - Нередко вместо термина "набор символов" неправильно употребляют термин **"кодовая страница" (code page)**, означающий на самом деле частный случай набора символов с однобайтным кодированием

- **Символ** (character) — это минимальная единица текста, имеющая семантическую ценность.
- **Кодированный набор символов (coded character set)** — это набор символов, в котором каждому **символу соответствует уникальный номер**.
- **Код символа в наборе символов (кодовой таблице) (code point of a coded character set)** — это любое допустимое значение в наборе символов или кодовом пространстве.
- **Кодовое пространство (code space)** — диапазон целых чисел, значениями которых являются кодовые точки.
- **code unit** — это «размер слова» схемы кодирования символов, например 7-битный, 8-битный, 16-битный. В некоторых схемах некоторые символы кодируются с использованием нескольких единиц кода, что приводит к кодированию переменной длины.

**Распространенные кодировки**

В настоящее время чаще всего используются кодировки трёх типов:

- **ASCII и совместимые с ней**
- **основанные на Юникоде (Unicode)**
- (реже) совместимые с EBCDIC
- в мире существует огромное количество различных кодировок (только для русского языка есть 4 кодировки, кроме вариантов, основанных на Unicode).

В программах с использованием русского языка используются такие варианты:

- **Кодировки, основанные на Юникоде (Unicode)**.
- **Кодировка CP1251 (Windows-1251)** — используется, если необходимо иметь русские символы непрерывным массивом для лёгкости их обработки, и в случае использования ОС Windows, перекодировать такой текст можно без использования стороннего ПО.
- Для русского языка еще существуют кодировки: CP-866, KOI-8R, ISO-8859-5.

**Кодировка ASCII**

**Телетайп - электромеханическая печатная машина, используемая для передачи между двумя абонентами текстовых сообщений по электрическому каналу**

ASCII (American standard code for information interchange) — название кодировки (набора, таблицы) символов, в которой некоторым распространённым печатным и непечатным символам сопоставлены числовые коды.

- Длинна кодовой таблицы ASCII - 128 символов (от 0 до 127) или 7 бит.
- Таблица была разработана и стандартизирована в США, в 1963 году и первоначально использовалась для телетайпов (которые использовались для передачи текстовых сообщений и были первыми терминалами для цифровых вычислительных машин).

**Кодовая таблица ASCII**

*USASCII code chart*

| $b_7$ $b_6$ $b_5$ → / Bits ↓ | | | | 0 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1 0 0 | 1 0 1 | 1 1 0 | 1 1 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $b_4$ | $b_3$ | $b_2$ | $b_1$ | Row ↓ / Column → | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 0 | 0 | 0 | 0 | NUL | DLE | SP | 0 | @ | P | ` | p |
| 0 | 0 | 0 | 1 | 1 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 0 | 0 | 1 | 0 | 2 | STX | DC2 | " | 2 | B | R | b | r |
| 0 | 0 | 1 | 1 | 3 | ETX | DC3 | # | 3 | C | S | c | s |
| 0 | 1 | 0 | 0 | 4 | EOT | DC4 | $ | 4 | D | T | d | t |
| 0 | 1 | 0 | 1 | 5 | ENQ | NAK | % | 5 | E | U | e | u |
| 0 | 1 | 1 | 0 | 6 | ACK | SYN | & | 6 | F | V | f | v |
| 0 | 1 | 1 | 1 | 7 | BEL | ETB | ' | 7 | G | W | g | w |
| 1 | 0 | 0 | 0 | 8 | BS | CAN | ( | 8 | H | X | h | x |
| 1 | 0 | 0 | 1 | 9 | HT | EM | ) | 9 | I | Y | i | y |
| 1 | 0 | 1 | 0 | 10 | LF | SUB | * | : | J | Z | j | z |
| 1 | 0 | 1 | 1 | 11 | VT | ESC | + | ; | K | [ | k | { |
| 1 | 1 | 0 | 0 | 12 | FF | FS | , | < | L | \ | l | \| |
| 1 | 1 | 0 | 1 | 13 | CR | GS | - | = | M | ] | m | } |
| 1 | 1 | 1 | 0 | 14 | SO | RS | . | > | N | ^ | n | ~ |
| 1 | 1 | 1 | 1 | 15 | SI | US | / | ? | O | _ | o | DEL |

**Кодовая таблица ASCII**

Таблица ASCII определяет коды для символов:

- управляющих символов
- десятичных цифр
- латинского алфавита (заглавных и строчных букв)
- знаков препинания
- (в специальных версиях) национального алфавита

In [185]:
```
# коды цифр, символов латинского алфавита ASCII удобно подобраны для чтения в двоичном ф
for s in ['0', '1', '2', '...', '9', '------\n',
          'A', 'B', 'C', '...', 'Y', 'Z', '------\n',
          'a', 'b', 'c', '...', 'y', 'z']:
    if len(s) == 1:
        print(f'{s}: decimal {ord(s):3} hex 0x{ord(s):02x} binary {ord(s):08b}')
    else:
        print(s)
```

```
0: decimal  48 hex 0x30 binary 00110000
1: decimal  49 hex 0x31 binary 00110001
2: decimal  50 hex 0x32 binary 00110010
...
9: decimal  57 hex 0x39 binary 00111001
------

A: decimal  65 hex 0x41 binary 01000001
B: decimal  66 hex 0x42 binary 01000010
C: decimal  67 hex 0x43 binary 01000011
...
Y: decimal  89 hex 0x59 binary 01011001
Z: decimal  90 hex 0x5a binary 01011010
------

a: decimal  97 hex 0x61 binary 01100001
b: decimal  98 hex 0x62 binary 01100010
c: decimal  99 hex 0x63 binary 01100011
...
y: decimal 121 hex 0x79 binary 01111001
z: decimal 122 hex 0x7a binary 01111010
```

**Управляющие коды ASCII**

| Номер | Английское название | Русское название | Соче-тание клавиш | Escape последовательность | Назначение |
|---|---|---|---|---|---|
| 0x00 | NULL | пустой символ | ^@ | \0 | Этот символ ничего не делает. Некоторые терминалы изображают его как пробел, но это неправильно. Часто NULL используют для обозначения конца цепочки символов (например, в языке C). |
| 0x03 | END OF TEXT | конец текста | ^C | | При вводе на терминале обычно интерпретируется как сигнал прерывания. |
| 0x04 | END OF TRANSMISSION | конец передачи | ^D | | При вводе на терминале в UNIX-системах интерпретируется как конец вводимых данных. Если текущая программа брала данные с терминала, то она завершается, как только обработает всё, что было до символа ^D. |
| 0x08 | BACKSPACE | возврат на шаг | ^H | \b | Перемещает позицию печати на один символ назад. На принтерах может использоваться для наложения одного символа на другой, например a BS ^ = â. При вводе с терминала иногда используется для стирания предшествующего символа («забой»). |
| 0x09 | CHARACTER TABULATION (horizontal tabulation) | горизонтальная табуляция | ^I | \t | Перемещает позицию печати к следующей позиции горизонтальной табуляции. |
| 0x0A | LINE FEED | перевод строки | ^J | \n | Перемещает позицию печати на одну строку вниз (исходно — без *возврата каретки*) . Разделяет строки текстовых файлов в Unix-системах. |
| 0x0B | LINE TABULATION (vertical tabulation) | вертикальная табуляция | ^K | \v | Перемещает позицию печати к следующей позиции вертикальной табуляции. На терминалах этот символ обычно эквивалентен переводу строки. |
| 0x0D | CARRIAGE RETURN | Возврат каретки | ^M | \r | Перемещает позицию печати в крайнее левое положение (исходно — без *перевода на следующую строку*). Разделяет строки текстовых файлов в некоторых ОС (например Mac OS, но не в Mac OS X). Во многих других ОС (CP/M, MS-DOS и Microsoft Windows), для разделения строк используется сочетание кодов возврата каретки (CARRIAGE RETURN) и перевода строки (LINE FEED), то есть в том виде, в котором файл можно отправить непосредственно на принтер. |
| 0x0E | SHIFT OUT (locking-shift one) | режим национальных символов | ^N | | В КОИ-7 включает режим национальных символов. На некоторых принтерах включает режим символов двойной ширины. |
| 0x0F | SHIFT IN (locking-shift zero) | режим обычного ASCII | ^O | | В КОИ-7 включает латинский режим. На некоторых принтерах включает режим узких символов. |
| 0x1A | SUBSTITUTE | символ замены | ^Z | | Ставится на месте символов, значения которых были потеряны при передаче. В CP/M и MS-DOS использовался для обозначения конца текстовых файлов и конца вводимых с консоли данных (хотя для этого были предназначены символы ^C и ^D). |
| 0x1B | ESCAPE | Альтернативный регистр № 2 (АР2) | ^[ | \e | Означает, что следующие за ним символы имеют какое-то другое значение, отличное от того, которое определено в ASCII. Обычно начинает управляющие последовательности. См. также ANSI.SYS. |
| 0x7F | DELETE | удаление | ^? | | Предназначен для забивания ошибочно пробитых символов на семидорожечных перфолентах (поскольку обозначается пробитием дырочек во всех дорожках), поэтому там он эквивалентен пустому символу (\0). На терминалах может генерироваться нажатием либо кнопки Backspace, либо кнопки Delete. |

**Некоторые управляющие коды ASCII**

- В таблице ASCII было определено 29 управляющих символов, большинство из которых находилось в диапазоне 0x00h-0x1F.
- Управляющие символы ASCII предназначались для управления работой телетайпов и видеотерминалов и вводились на них сочетаниями с клавишей Ctrl, которая обнуляла в коде введённой клавиши бит 6. В современных компьютерных системах нигде, кроме эмуляторов терминала, не предусмотрен ввод этих символов напрямую (кроме символов табуляции и перевода строки) и большинство из перечисленных управляющих символов не используется.

### Переход на новую строку

Разные платформы (ОС) используют разные коды в качестве символа новой строки, в частности:

- Windows: `\r\n`
- Unix: `\n`
- Старые компьютеры Макинтош: `\r`
- Существуют системы использующие: `\n\r`

При открытии файла в текстовом режиме в Python 3 все варианты окончания строки будут автоматически конвертироваться к `\n` (как при записи, так и при чтении). Например, в этом режиме в Windows при чтении символ `\r` будет удален, а при записи, наоборот, добавлен.

In [186]:
```
cd E:\YandexDisk\Python\Ipynb\TOBD_2022\03_data_files\lec\
```

```
E:\YandexDisk\Python\Ipynb\TOBD_2022\03_data_files\lec
```

In [188]:
```
with open("lec03_example.txt", "wt", encoding="cp1251") as f:
    print('Строка1\nСтрока2', file=f)
    print('Строка3', end='', file=f) # значение по умолчанию end='\n'
    print(' продолжение строки3', file=f)
```

In [189]:
```
with open("lec03_example.txt", "rt", encoding="cp1251") as file:
    for line in file:
        print(line, end='')
```

```
Строка1
Строка2
Строка3 продолжение строки3
```

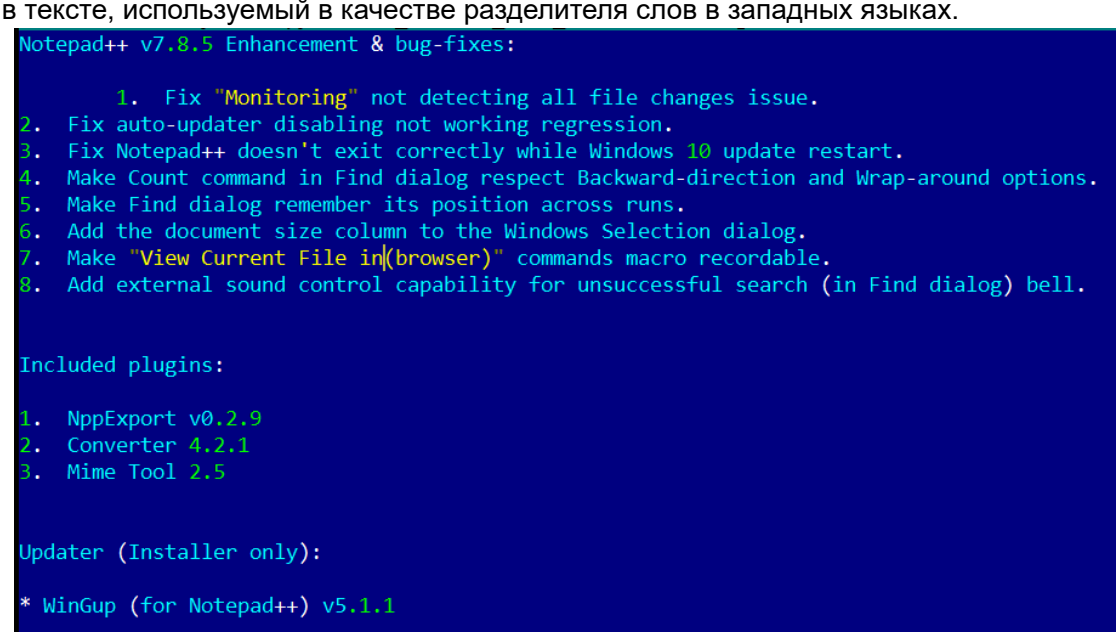
In [190]:
```
with open("lec03_example.txt", "rb") as file:
    for bs in file:
        print(bs)
        print(bs.decode("cp1251"))
```

```
b'\xd1\xf2\xf0\xee\xea\xe01\r\n'
Строка1

b'\xd1\xf2\xf0\xee\xea\xe02\r\n'
Строка2

b'\xd1\xf2\xf0\xee\xea\xe03 \xef\xf0\xee\xe4\xee\xeb\xe6\xe5\xed\xe8\xe5 \xf1\xf2\xf0
\xee\xea\xe83\r\n'
Строка3 продолжение строки3
```

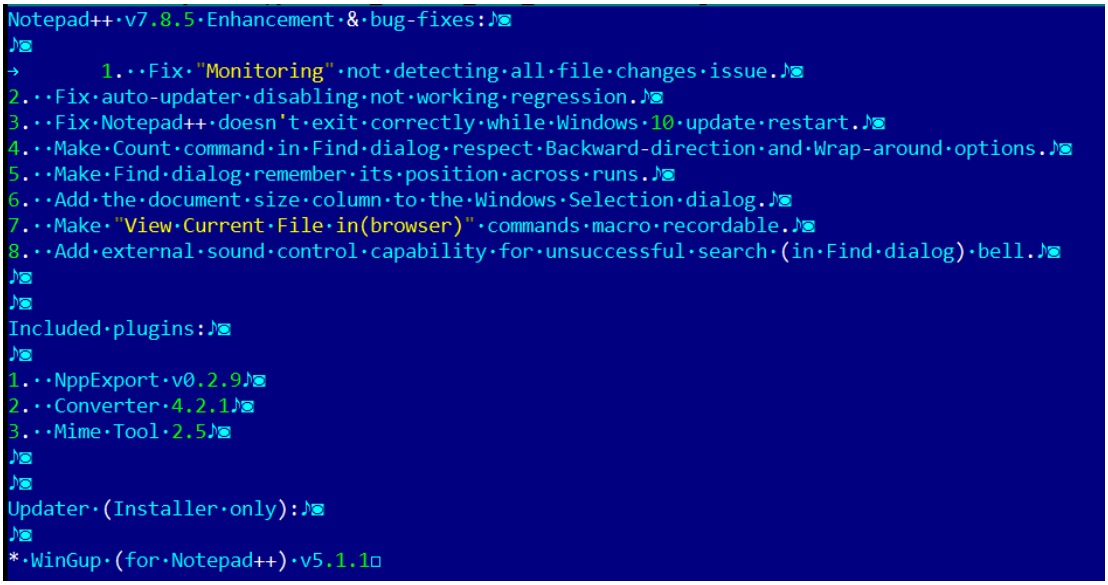
### Пробельные символы

**Пробельные символы** (Whitespace character) - символ или серия символов, которые представляют горизонтальное или вертикальное пространство в типографике. При отображении пробельные символы не соответствует видимогу символу, но обычно занимают область на отображаемой странице.

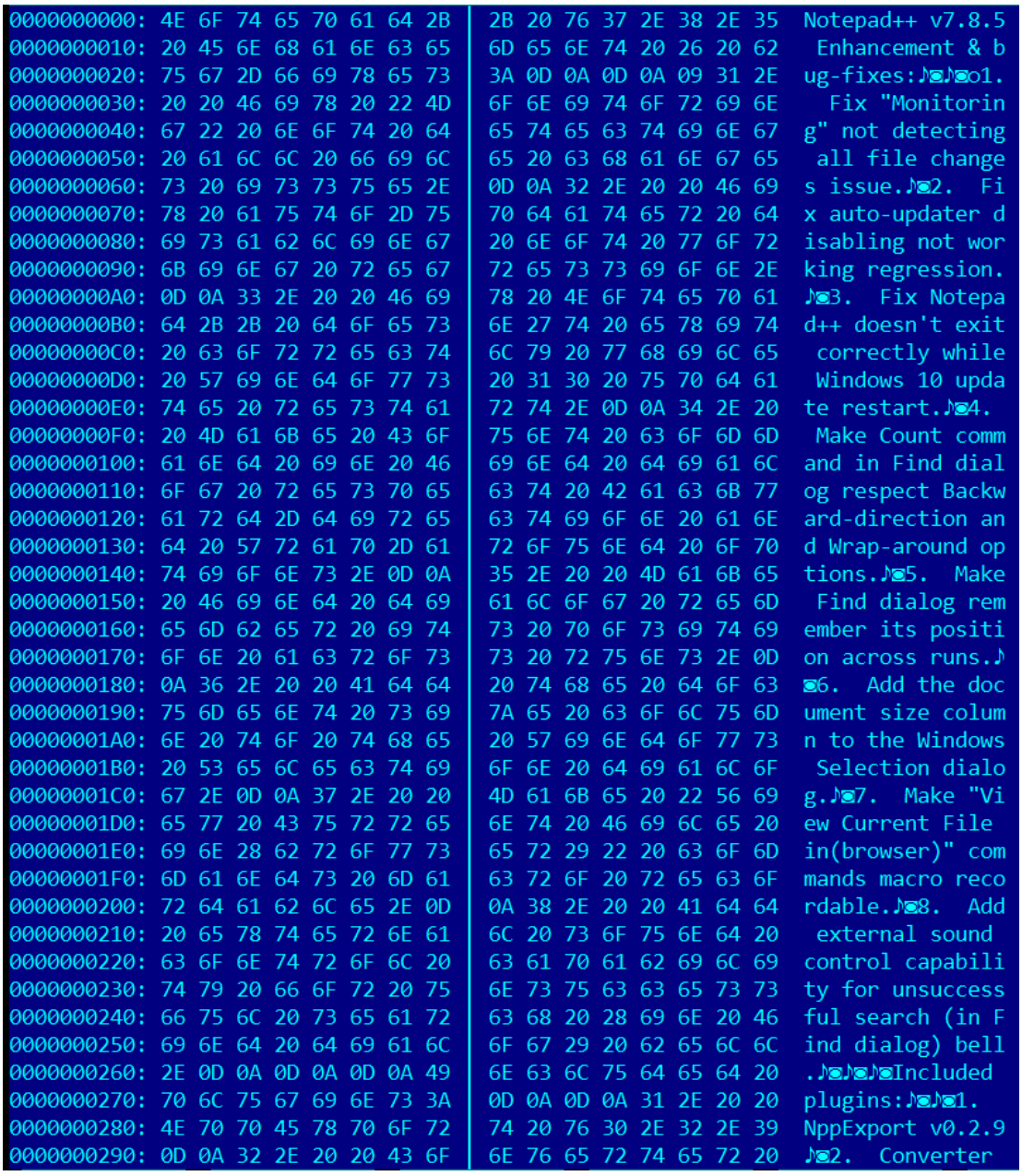
Например, символ проблеа U+0020 (также ASCII 32) (в Питоне: `' '`) представляет собой пустой символ в тексте, используемый в качестве разделителя слов в западных языках.




**Пример текстового файла при просмотре в обычном редакторе**




**Пример текстового файла с демонстрацией пробельных символов**




**Пример фрагмента текстового файла в режиме побайтного просмотра**

### Кодировка Windows 1251

**Windows-1251** — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии.

- Синонимы: CP1251; ANSI (только в русскоязычной ОС Windows).

- Выгодно отличается от других 8-битных кириллических кодировок (таких как CP866, KOI8-R и ISO 8859-5) наличием практически всех символов, использующихся в русской типографике для обычного текста (отсутствует только значок ударения).
- Содержит все символы для других славянских языков: украинского, белорусского, сербского, македонского и болгарского.
- Первая половина таблицы кодировки (коды от 0x00 до 0x7F) полностью соответствует кодировке ASCII

| символ | 10-й код | 2-й код | символ | 10-й код | 2-й код | символ | 10-й код | 2-й код | символ | 10-й код | 2-й код |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Ђ | 128 | 10000000 |  | 160 | 10100000 | А | 192 | 11000000 | а | 224 | 11100000 |
| Ѓ | 129 | 10000001 | Ў | 161 | 10100001 | Б | 193 | 11000001 | б | 225 | 11100001 |
| ‚ | 130 | 10000010 | ў | 162 | 10100010 | В | 194 | 11000010 | в | 226 | 11100010 |
| ѓ | 131 | 10000011 | Ј | 163 | 10100011 | Г | 195 | 11000011 | г | 227 | 11100011 |
| „ | 132 | 10000100 | ¤ | 164 | 10100100 | Д | 196 | 11000100 | д | 228 | 11100100 |
| … | 133 | 10000101 | Ґ | 165 | 10100101 | Е | 197 | 11000101 | е | 229 | 11100101 |
| † | 134 | 10000110 | ¦ | 166 | 10100110 | Ж | 198 | 11000110 | ж | 230 | 11100110 |
| ‡ | 135 | 10000111 | § | 167 | 10100111 | З | 199 | 11000111 | з | 231 | 11100111 |
| € | 136 | 10001000 | Ё | 168 | 10101000 | И | 200 | 11001000 | и | 232 | 11101000 |
| ‰ | 137 | 10001001 | © | 169 | 10101001 | Й | 201 | 11001001 | й | 233 | 11101001 |
| Љ | 138 | 10001010 | Є | 170 | 10101010 | К | 202 | 11001010 | к | 234 | 11101010 |
| ‹ | 139 | 10001011 | « | 171 | 10101011 | Л | 203 | 11001011 | л | 235 | 11101011 |
| Њ | 140 | 10001100 | ¬ | 172 | 10101100 | М | 204 | 11001100 | м | 236 | 11101100 |
| Ќ | 141 | 10001101 | - | 173 | 10101101 | Н | 205 | 11001101 | н | 237 | 11101101 |
| Ћ | 142 | 10001110 | ® | 174 | 10101110 | О | 206 | 11001110 | о | 238 | 11101110 |
| Џ | 143 | 10001111 | Ї | 175 | 10101111 | П | 207 | 11001111 | п | 239 | 11101111 |
| ђ | 144 | 10010000 | ° | 176 | 10110000 | Р | 208 | 11010000 | р | 240 | 11110000 |
| ‘ | 145 | 10010001 | ± | 177 | 10110001 | С | 209 | 11010001 | с | 241 | 11110001 |
| ’ | 146 | 10010010 | І | 178 | 10110010 | Т | 210 | 11010010 | т | 242 | 11110010 |
| “ | 147 | 10010011 | і | 179 | 10110011 | У | 211 | 11010011 | у | 243 | 11110011 |
| ” | 148 | 10010100 | ґ | 180 | 10110100 | Ф | 212 | 11010100 | ф | 244 | 11110100 |
| • | 149 | 10010101 | µ | 181 | 10110101 | Х | 213 | 11010101 | х | 245 | 11110101 |
| – | 150 | 10010110 | ¶ | 182 | 10110110 | Ц | 214 | 11010110 | ц | 246 | 11110110 |
| — | 151 | 10010111 | · | 183 | 10110111 | Ч | 215 | 11010111 | ч | 247 | 11110111 |
| □ | 152 | 10011000 | ё | 184 | 10111000 | Ш | 216 | 11011000 | ш | 248 | 11111000 |
| тм | 153 | 10011001 | № | 185 | 10111001 | Щ | 217 | 11011001 | щ | 249 | 11111001 |
| љ | 154 | 10011010 | є | 186 | 10111010 | Ъ | 218 | 11011010 | ъ | 250 | 11111010 |
| › | 155 | 10011011 | » | 187 | 10111011 | Ы | 219 | 11011011 | ы | 251 | 11111011 |
| њ | 156 | 10011100 | ј | 188 | 10111100 | Ь | 220 | 11011100 | ь | 252 | 11111100 |
| ќ | 157 | 10011101 | Ѕ | 189 | 10111101 | Э | 221 | 11011101 | э | 253 | 11111101 |
| ћ | 158 | 10011110 | ѕ | 190 | 10111110 | Ю | 222 | 11011110 | ю | 254 | 11111110 |
| џ | 159 | 10011111 | ї | 191 | 10111111 | Я | 223 | 11011111 | я | 255 | 11111111 |

**Кодовая таблица Windows 1251 (вторая половина)**

Особенности (недостатки) Windows-1251:

- Строчная буква «я» имеет код 0xFF (255 в десятичной системе), что в некоторых случаях может приводить к техническим проблемам.
- Отсутствуют символы псевдографики, имеющиеся в CP866 и KOI8.
- Символы 'Ё' и 'ё' находятся вне алфавитной полседовательности.

In [191]:
```
alph = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
for s, b in zip(alph, bytes(alph, encoding='cp1251')):
    print(f'{s}: {b:08b}, int: {int(b)}')
```

```
а: 11100000, int: 224
б: 11100001, int: 225
в: 11100010, int: 226
г: 11100011, int: 227
д: 11100100, int: 228
е: 11100101, int: 229
ё: 10111000, int: 184
ж: 11100110, int: 230
з: 11100111, int: 231
и: 11101000, int: 232
й: 11101001, int: 233
к: 11101010, int: 234
л: 11101011, int: 235
м: 11101100, int: 236
н: 11101101, int: 237
о: 11101110, int: 238
п: 11101111, int: 239
р: 11110000, int: 240
с: 11110001, int: 241
т: 11110010, int: 242
у: 11110011, int: 243
ф: 11110100, int: 244
х: 11110101, int: 245
ц: 11110110, int: 246
ч: 11110111, int: 247
ш: 11111000, int: 248
щ: 11111001, int: 249
ъ: 11111010, int: 250
ы: 11111011, int: 251
ь: 11111100, int: 252
э: 11111101, int: 253
ю: 11111110, int: 254
я: 11111111, int: 255
```

## Unicode


- к оглавлению


#### Юникод

**Юникод (Unicode)** — стандарт кодирования символов, включающий в себя знаки почти всех письменных языков мира. В настоящее время стандарт является преобладающим в Интернете.

Стандарт состоит из двух основных частей:

- **Универсального набора символов (Universal character set, UCS)** который перечисляет допустимые по стандарту Юникод символы и присваивает каждому символу код в виде неотрицательного целого числа, записываемого обычно в шестнадцатеричной форме с префиксом U+, например, U+040F.
- Семейства **кодировок (Unicode transformation format, UTF)** которое определяет способы преобразования кодов символов для передачи в потоке или в файле.

Причины создания юникода:

- Проблема неправильной раскодировки - вызвана отсутствием стандартизированной формы указания кодировки для файла и могла быть решена внедрением общей для всех языков

кодировки.
- Проблема ограниченности набора символов - проблему можно было решить либо переключением шрифтов внутри документа, либо внедрением «широкой», универсальной кодировки.
- Проблема преобразования одной кодировки в другую - требовала точного понимания исходной и конечной кодировки и таблиц преобразования между ними.
- Проблема дублирования шрифтов - для каждой кодировки создавался свой шрифт, даже если наборы символов в кодировках совпадали, можно было бы решить созданием "больших" шрифтов, покрывающих самые различные символы, но это требовало создания единого реестра символов.

Была признана необходимость создания единой «широкой» (универсальной) кодировки. Кодировки с переменной длиной символа, широко использующиеся в Восточной Азии, были признаны слишком сложными и в первых версях юникода (с 1991 г.) было решено использовать 16-битные символы фиксированной ширины, так как 8-битных символов очевидно нехватало, а использование 32-битных символов казалось слишком расточительным.

**Универсальный набор символов и плоскости Юникод**

- Первая версия Юникода представляла собой кодировку с фиксированным размером символа в 16 бит, то есть общее число кодов было 2^16 (65 536 значений).
  - При этом в Юникоде планировалось кодировать не все существующие символы, а только те, которые необходимы в повседневном обиходе. Редко используемые символы должны были размещаться в «области пользовательских символов» (private use area), которая первоначально занимала коды U+D800…U+F8FF.
  - Чтобы использовать Юникод также и в качестве промежуточного звена при преобразовании разных кодировок друг в друга, в него включили все символы, представленные во всех наиболее известных кодировках.
- В дальнейшем, однако, было принято решение кодировать все символы и проводить политику активного расширения набора символов, например, включив в него не только символы языков но и эмодзи. В связи с этим потребовалось значительно расширить кодовую область.
  - В связи с этим коды символов стали рассматриваться не как 16-битные значения, а как абстрактные числа универсального набора символов, которые в компьютере могут представляться множеством разных способов.
  - В текущей версии Unicode 15.0 описано 149 186 симовлов (в версии 1.1 их было всего 34 233).
  - Для текущего универсального набора символов недостаточно первоначальных 16 бит, сейчас в Unicode зарезервировано 1 112 064 кодов символов (для кодирования с избытком достаточно 21 бита, используется подмножество 21 битных значений, совместимое с форматом UTF-16).

Символы универсального набора символов Unicode обозначаются следующим образом:

- Для обозначения символов Unicode используется запись вида «U+xxxx» (для кодов 0…FFFF) где xxx — шестнадцатеричные цифры.
  - Например, символ «я» (U+044F) имеет код 044F16 = 110310.
- Для кодов 10000…FFFFF: «U+xxxxx» и для кодов 100000…10FFFF «U+xxxxxx».

Кодовое пространство разбито на **17 плоскостей (planes)** по 2^16 (65 536) символов.

- Нулевая плоскость (plane 0) называется базовой (basic) и содержит символы наиболее употребительных письменностей.
- Остальные плоскости — дополнительные (supplementary).


**Базовая плоскость универсального набора символов**

В Юникоде 14.0 в базовой плоскости представлены следующие блоки:

- **Основная латиница (0000—007F)** - точное повторение таблицы ASCII, что, в частности, позволяет фортмату UTF-8 быть обратно совестимым с кодировкой ASCII.
- Дополнение к латинице — 1 (0080—00FF)
- Расширенная латиница — A (0100—017F)
- Расширенная латиница — B (0180—024F)
- Расширения МФА (0250—02AF)
- Модификаторы букв (02B0—02FF)
- Комбинируемые диакритические знаки (0300—036F)
- Греческое и коптское письмо (0370—03FF)
- **Кириллица (0400—04FF)**, см. также Кириллица в Юникоде
  - Существует неоднозначность по отношению к кодированию некоторых букв: например, «Й» может быть закодирована как единый символ U+0419 либо как комбинация «И» U+0418 и диакритического знака U+0306.
- ...

Блок содержит все буквы и управляющие коды из кодировки ASCII.

In [119]:
```
# Буквы русского алфавита в UCS:
alph = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
for s in alph:
    bs = bytes(s, encoding='UTF-16-BE') # обратите внимание на кодировку!
    s_i = int.from_bytes(bs, byteorder='little', signed=False)
    print(f'{s}: U-{bs[0]:02x}{bs[1]:02x} {bs[0]:08b} {bs[1]:08b}, int: {s_i}')
```

```
а: U-0430 00000100 00110000, int: 12292
б: U-0431 00000100 00110001, int: 12548
в: U-0432 00000100 00110010, int: 12804
г: U-0433 00000100 00110011, int: 13060
д: U-0434 00000100 00110100, int: 13316
е: U-0435 00000100 00110101, int: 13572
ё: U-0451 00000100 01010001, int: 20740
ж: U-0436 00000100 00110110, int: 13828
з: U-0437 00000100 00110111, int: 14084
и: U-0438 00000100 00111000, int: 14340
й: U-0439 00000100 00111001, int: 14596
к: U-043a 00000100 00111010, int: 14852
л: U-043b 00000100 00111011, int: 15108
м: U-043c 00000100 00111100, int: 15364
н: U-043d 00000100 00111101, int: 15620
о: U-043e 00000100 00111110, int: 15876
п: U-043f 00000100 00111111, int: 16132
р: U-0440 00000100 01000000, int: 16388
с: U-0441 00000100 01000001, int: 16644
т: U-0442 00000100 01000010, int: 16900
у: U-0443 00000100 01000011, int: 17156
ф: U-0444 00000100 01000100, int: 17412
х: U-0445 00000100 01000101, int: 17668
ц: U-0446 00000100 01000110, int: 17924
ч: U-0447 00000100 01000111, int: 18180
ш: U-0448 00000100 01001000, int: 18436
щ: U-0449 00000100 01001001, int: 18692
ъ: U-044a 00000100 01001010, int: 18948
ы: U-044b 00000100 01001011, int: 19204
ь: U-044c 00000100 01001100, int: 19460
э: U-044d 00000100 01001101, int: 19716
ю: U-044e 00000100 01001110, int: 19972
я: U-044f 00000100 01001111, int: 20228
```

In [192]:
```
# запись символа й:
print("U-0439:\u0439")
# запись символа й как и с диакритическим знаком:
print('U-0438 U-0306:\u0438\u0306')
# запись символа по названию в Unicode: https://ru.wikipedia.org/wiki/Кириллица_в_Юникоd
print('cyrillic small letter short i:\N{cyrillic small letter short i}')
```

```
U-0439:й
U-0438 U-0306:й
cyrillic small letter short i:й
```

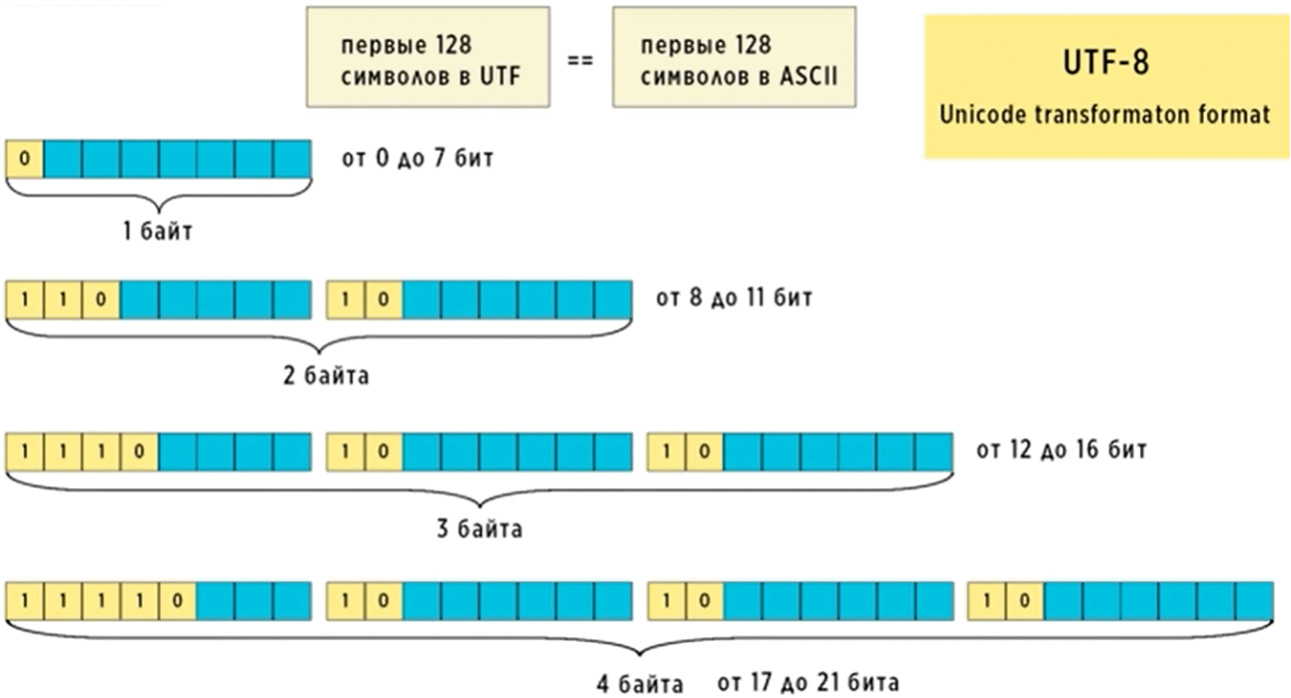
In [193]:
```
# Пример использования unicode для работы с эмоджи в Python:
print("\U0001f600")
print("\N{grinning face}")
```

😀
😀

**UTF-8**

**UTF-8 (Unicode Transformation Format)** — распространённый стандарт кодирования символов, позволяющий компактно хранить и передавать символы Юникода, используя переменное количество байт (от 1 до 4).

- UTF-8 беспечивает **полную обратную совместимость с 7-битной кодировкой ASCII**.
- UTF-8 сейчас **является доминирующей в веб-пространстве** и широко распространена в UNIX-подобных операционных системах.
- UTF-8, по сравнению с UTF-16, **даёт наибольший выигрыш в компактности для текстов на латинице**, поскольку латинские буквы без диакритических знаков, цифры и наиболее распространённые знаки препинания кодируются в UTF-8 лишь одним байтом и коды этих символов соответствуют их кодам в ASCII.
- Идентификатор кодировки в Windows — 65001.




**Кодирование символов Unicode в переменном количестве байт**

| Символ | | Двоичный код символа | UTF-8 в двоичном виде | UTF-8 в шестнадцатеричном виде |
|---|---|---|---|---|
| $ | U+0024 | 0100100 | 00100100 | 24 |
| ¢ | U+00A2 | 10100010 | 11000010 10100010 | C2 A2 |
| € | U+20AC | 100000 10101100 | 11100010 10000010 10101100 | E2 82 AC |
| 𐍈 | U+10348 | 1 00000011 01001000 | 11110000 10010000 10001101 10001000 | F0 90 8D 88 |

**Пример кодирование символов Unicode в переменном количестве байт UTF-8**

Для указания, что файл содержит символы Юникода, в начале файла или может быть вставлен маркер последовательности байтов UTF-8: 0xEF 0xBB 0xBF

Не всякая последовательность байтов является допустимой. Декодер UTF-8 должен понимать и адекватно обрабатывать такие ошибки:

- Недопустимый байт.
- Байт продолжения (10xxxxxx) без начального байта.
- Отсутствие нужного количества байтов продолжения 10xxxxxx — например, двух после 1110xxxx).
- Строка обрывается посреди символа.
- Неэкономное кодирование — например, кодирование символа тремя байтами, когда можно двумя. (Существует нестандартный вариант UTF-8, который кодирует символ с кодом 0 как 1100.0000 1000.0000, отличая его от символа конца строки 0000.0000.)
- Последовательность байтов, декодирующаяся в недопустимую кодовую позицию (например символы суррогатных пар UTF-16).

**UTF-16**

UTF-16 (Unicode Transformation Format) — способов кодирования символов из Юникода в виде последовательности 16-битных слов (по 2 байта).

- Данная кодировка позволяет записывать символы Юникода в диапазонах U+0000..U+D7FF и U+E000..U+10FFFF (общим количеством 1 112 064).
- При этом каждый символ записывается одним или двумя словами (суррогатная пара).
- Исключенный отсюда диапазон D80016..DFFF16 используется для кодирования так называемых суррогатных пар.
- В первой версии Юникод (1991 г.) представляет собой 16-битную кодировку с фиксированной шириной символа (UCS-2) во второй версии был добавлен UTF-16 для расширения кодируемого диапазона до 1 112 064 символов.

Один символ кодировки UTF-16 представлен **последовательностью из двух байтов** или двух пар байтов. Какой из двух байтов идёт впереди, старший или младший, зависит от порядка байтов:

- **little endian** - порядок байтов от старшего к младшему $B_0, B_1, \ldots, B_{N-1}$ (принят в процессорах x86) Систему, совместимую с процессорами x86, называют little endian,
- **big endian** - порядок байтов от младшегок к старшему $B_{N-1}, \ldots, B_1, B_0$ (принят в процессорах SPARC)
- Для определения порядка байтов используется **метка порядка байто (byte order mark)**. В начале текста записывается код U+FEFF.
- При считывании, если вместо U+FEFF считалось U+FFFE, значит порядок байтов обратный (little endian).
- Поскольку код U+FFFE в Юникоде не кодирует символ и зарезервирован как раз для целей определения порядка байтов.
- Так как в кодировке UTF-8 не используются значения 0xFE и 0xFF, можно использовать метку порядка байтов как признак, позволяющий различать UTF-16 и UTF-8.

In [194]:
```
# Примеры работы различных кодировок UTF:
print('\u0430')
alph = 'a'
enc_l = ['UTF-8', 'UTF-16', 'UTF-16-BE', 'UTF-16-LE']
for enc in enc_l:
    bs = bytes('a', encoding=enc)
    print(f'{enc}:',[f'0x{b:08b}' for b in bs])
    print(f'{enc}:',[f'0x{b:02x}' for b in bs],'\n')
```

```
a
UTF-8: ['0x11010000', '0x10110000']
UTF-8: ['0xd0', '0xb0']

UTF-16: ['0x11111111', '0x11111110', '0x00110000', '0x00000100']
UTF-16: ['0xff', '0xfe', '0x30', '0x04']

UTF-16-BE: ['0x00000100', '0x00110000']
UTF-16-BE: ['0x04', '0x30']

UTF-16-LE: ['0x00110000', '0x00000100']
UTF-16-LE: ['0x30', '0x04']
```

**UTF-32**

**UTF-32 (Unicode Transformation Format)** — способ кодирования символов Юникода, использующий для кодирования любого символа ровно 32 бита.

- Символ UTF-32 является прямым представлением его кодовой позиции (Code point (англ.)рус.).

- Остальные кодировки, UTF-8 и UTF-16, используют для представления символов переменное число байтов.
- **(+)** Главное преимущество UTF-32 перед кодировками переменной длины заключается в том, что символы Юникод непосредственно индексируемы. Получение n-ой кодовой позиции является операцией, занимающей одинаковое время.
- **(-)** Главный недостаток UTF-32 — это неэффективное использование пространства, так как для хранения символа используется четыре байта.

UTF-32 применяется, главным образом, не в строках символов, а во внутренних API, где данные являются единственной кодовой позицией или глифом. Например, при прорисовке текста на последнем шаге происходит построение списка структур, каждая из которых включает в себя позиции x и y, атрибуты и единственный символ UTF-32, идентифицирующий глиф для прорисовки. Часто в «неиспользуемых» 11 битах каждого 32-битного символа хранится посторонняя информация.

- В программах на Python с версии 3.3 строки хранятся в UTF-32, но лидирующие нули оптимизируются в случае их неиспользования.
- UTF-32 используются для хранения строк в Unix в том случае, когда тип wchar_t определён как 32-битный.
- В ОС Windows, в которой тип wchar_t имеет размер 16 бит, строки UTF-32 почти не используются.

# Работа с бинарными файлами

- к оглавлению

#### Байтовые строки в Python

Базовая работа с бинарными файлами основана на манипулировании последовательностями байтов. Для этого в Python емеется тип `bytes` - неизменяемый тип, похожий на строки, но состоящий из целых чисел от 0 до 255 и его изменяемый аналог `bytesarray` .

In [440]:
```
# Примеры создания объектов bytes:

# заданное количество нулевых байтов:
bs0 = bytes(8)
print(bs0)

# На основе последовательности байтов в виде двух шестнадцетеричных цифр:
bs1 = b'\x62\x79\x74\x65\x73\x20\xd1'
print(f'bs1:{bs1}')

# На основе строки из символов ASCII:
bs2 = b'bytes' # тут допустимы только симоволы ASCII (коды от 0 до 127)
print(bs2)

# На основе строки Python с указанием кодировки:
bs3 = bytes('байты', encoding = 'cp1251')
print(f'bs3:{bs3}')

bs4 = bytes('байты', encoding = 'utf-8')
print(f'bs4:{bs4}')

# из строки с последовательностью шестнадцеричных цифр (каждая пара - 1 байт):
bs5 = bytes.fromhex('627974657320d1')
print(f'bs5:{bs5}')

# На основе последовательности числовых значений (от 0 до 255):
bs6 = bytes([98, 121, 116, 101, 115, 32, 209])
print(f'bs6:{bs6}')

# На основе генератора числовых значений (от 0 до 255):
bs7 = bytes(range(97, 107))
print(f'bs7:{bs7}')
```

```
b'\x00\x00\x00\x00\x00\x00\x00\x00'
bs1:b'bytes \xd1'
b'bytes'
bs3:b'\xe1\xe0\xe9\xf2\xfb'
bs4:b'\xd0\xb1\xd0\xb0\xd0\xb9\xd1\x82\xd1\x8b'
bs5:b'bytes \xd1'
bs6:b'bytes \xd1'
bs7:b'abcdefghij'
```

In [441]:
```
# оперирование элементами bytes происходит как с int:
print(type(bs1[0]), bs1[0])
```

```
<class 'int'> 98
```

In [442]:
```
# итерация по bytes возвращает значения типа int:
print(list(bs1))
```

```
[98, 121, 116, 101, 115, 32, 209]
```

In [443]:
```
# итерация по bytes возвращает значения типа int:
for i in bs1:
    print(i, type(i))
```

```
98 <class 'int'>
121 <class 'int'>
116 <class 'int'>
101 <class 'int'>
115 <class 'int'>
32 <class 'int'>
209 <class 'int'>
```

In [444]:
```
# получение строки с последовательностью шестнадцетеричных цифр:
print(bs1.hex())
```

```
627974657320d1
```

In [445]:
```
# сериализация числа в bytes:
my_int = 420420
my_int_bts_big = my_int.to_bytes(4, byteorder='big') # 4 - количество байт
my_int_bts_big
```

Out[445]:
```
b'\x00\x06jD'
```

In [446]:
```
# вариант сериализации с обратным порядком байтов:
my_int.to_bytes(4, byteorder='little')
```

Out[446]:
```
b'Dj\x06\x00'
```

In [447]:
```
# восстановление (десериализация данных из bytes):
int.from_bytes(my_int_bts_big, byteorder='big')
```

Out[447]:
```
420420
```

Тип bytes - неизменяемый, при необходимости изменять массив байтов необходимо использовать bytesarray.

In [448]:
```
# ошибка - bytes неизменяемый объект!
bs1[1] = 97
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-448-b8c724d944ec> in <module>
      1 # ошибка - bytes неизменяемый объект!
----> 2 bs1[1] = 97

TypeError: 'bytes' object does not support item assignment
```

In [449]:
```
bsa1 = bytearray(bs1)
bsa1
```

Out[449]:
```
bytearray(b'bytes \xd1')
```

In [450]:
```
bsa1[1]
```

Out[450]:
```
121
```

In [451]:
```
# bytesarray изменяемый объект:
bsa1[1] = 97
bsa1
```

Out[451]:
```
bytearray(b'bates \xd1')
```

In [453]:
```
# bytesarray изменяемый объект:
bsa1.append(97)
bsa1
```

Out[453]:
```
bytearray(b'bates \xd1aa')
```

Работа с бинарными файлами в Python:

In [454]:
```
numbers = [2, 4, 6, 8, 10, 12, 14]
numbers_bs = bytes(numbers)
numbers_bs
```

Out[454]:
```
b'\x02\x04\x06\x08\n\x0c\x0e'
```

In [455]:
```
# запись в bytes в бинарный файл:
# старый способ работы с файлом (нужно избегать!):
f_wb = open("numbers_v1.bin","wb")
try:
    f_wb.write(numbers_bs)
# без использования блока with необходимо вручную закрывать файл:
finally:
    f_wb.close()
```

In [456]:
```
# Чтение всего файла в виде одной строки байтов:
with open('numbers_v1.bin', 'rb') as f_rb1:
    content = f_rb1.read()

print(content)
print(list(content))
```

```
b'\x02\x04\x06\x08\n\x0c\x0e'
[2, 4, 6, 8, 10, 12, 14]
```

In [463]:
```
with open('numbers_v2.bin', 'wb') as f_wb3:
    for _ in range(3):
        int_chunk = [randint(1, 10_000_000) for _ in range(randint(2, 4))]
        print(int_chunk)
        f_wb3.write(b''.join(int_val.to_bytes(4, byteorder='big') for int_val in int_chu
```

```
[8381970, 623024, 7779606, 7657959]
[1694505, 4915883, 784382]
[470501, 8402757, 7723208]
```

In [464]:
```
import itertools

# Чтение файла блоками, с размером не больше заданного:
with open("numbers_v2.bin", "rb") as f_rb2:
    # читаем первый блок данных:
    bts = f_rb2.read(20) # 20 = 4 (байта на число) * 5 (чисел)
    while bts:
        print(f'len={len(bts)}: {bts}')
        print([int.from_bytes(bytes(chunk), byteorder='big')
               for chunk in itertools.zip_longest(*([iter(bts)] * 4))])
        # читаем очередной блок данных:
        bts = f_rb2.read(20)
```

```
len=20: b'\x00\x7f\xe6\x12\x00\t\x81\xb0\x00v\xb5\x16\x00t\xd9\xe7\x00\x19\xdb)'
[8381970, 623024, 7779606, 7657959, 1694505]
len=20: b'\x00K\x02\xab\x00\x0b\xf7\xfe\x00\x07-\xe5\x00\x807E\x00u\xd8\xc8'
[4915883, 784382, 470501, 8402757, 7723208]
```

# Сериализация и обмен данными

- к оглавлению

### Назначение различных форматов файлов

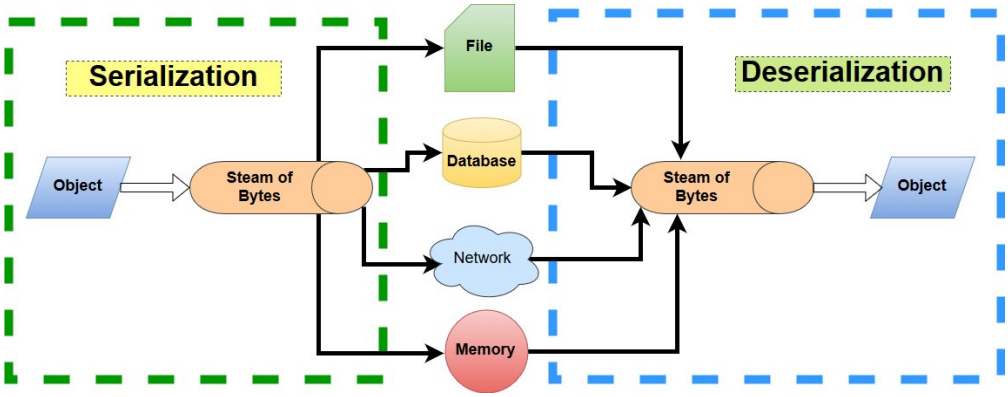
Форматы файлов для хранения данных можно разделить **по назначению форматов файлов**:

- **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** - обратимый перевод структуры данных в последовательность битов.

### Сериализация 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)




**Сериализация-десериализация данных**

Популярные форматы сериализации данных:

- Специализированные форматы рассмотриваемые в рамках курса:
  - **npy, npz** - стандартный бинарный формат двоичного файла в NumPy для сохранения массива / массивов NumPy. Формат npy разработан так, чтобы быть максимально простым при достижении ограниченных целей (рассмотрены на лекции 1).
  - **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.

- **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.

**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.

**Сравнение популярных форматов сериализации данных**

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.
- **YAML** (Yet Another Markup Language) - «дружественный» формат сериализации данных, концептуально близкий к языкам разметки, но ориентированный на удобство ввода-вывода типичных структур данных многих языков программирования, в том числе **удобство чтения и редактирование человеком**. Язык широко используется **для файлов конфигурации и в приложениях**, где данные хранятся или передаются. YAML нацелен на приложения, типичные для XML. YAML имеет **минимальный синтаксис**, который **намеренно отличается от SGML** и совместим с JSON т.к. использует `[...]` для списков и `{...}` для словарей. Для обозначения вложенности используются отступы, как в Python.

# 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.
- Pickle небезопасный формат сериализации так как не защищен от ошибочных или вредоносных данных.

Сохранение данных в Pickle

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

In [261]:
```
# объект (данные) для сохранения:
dat1 = ["Строка", (12, 3)]
```

In [262]:
```
import pickle # подключаем модуль pickle
```

In [263]:
```
# Стандартное расширение дл

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

Pickle хранит данные в бинарном формате:

In [264]:
```
with open('dat1.pickle') as f:
    for l in f:
        print(l)
```

```
Ђ ]q(XРЎС,СЪPsPєP°q KK †q e.
```

In [265]:
```
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\xd0\xb0q\x0
1K\x0cK\x03\x86q\x02e.'
```

Восстановление объектов (данных) из файла pickle:

In [266]:
```
with open('dat1.pickle', 'rb') as f:
    dat1_l = pickle.load(f)

dat1_l
```

Out[266]:
```
['Строка', (12, 3)]
```

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

In [267]:
```
dat2 = (6, 7, 8, 9, 10)
```

In [268]:
```
with open('dat2.pickle', 'wb') as f:
    pickle.dump(dat1, f)
    pickle.dump(dat2, f)
```

In [269]:
```
# 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[269]:
```
(['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 [270]:
```
bs = pickle.dumps(dat1)
bs
```

Out[270]:
```
b'\x80\x03]q\x00(X\x0c\x00\x00\x00\xd0\xa1\xd1\x82\xd1\x80\xd0\xbe\xd0\xba\xd0\xb0q\x01K\x0cK\x03\x86q\x02e.'
```

In [271]:
```
type(bs)
```

Out[271]:
```
bytes
```

In [272]:
```
pickle.loads(bs)
```

Out[272]:
```
['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 получившуюся 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:

```
open (<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 присутствует, 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 [273]:
```
import shelve
```

In [274]:
```
db = shelve.open("shl_1")
```

In [275]:
```
db["obj1"] = [1, 2, 3, 4, 5]
db["obj2"] = (6, 7, 8, 9, 10)
```

In [276]:
```
db["obj1"]
```

Out[276]:
```
[1, 2, 3, 4, 5]
```

In [277]:
```
db["obj2"]
```

Out[277]:
```
(6, 7, 8, 9, 10)
```

In [278]:
```
db.close()
```

In [279]:
```
db = shelve.open('shl_1')
```

In [280]:
```
db.keys()
```

Out[280]:
```
KeysView(<shelve.DbfilenameShelf object at 0x000002C15AB4A548>)
```

In [281]:
```
list(db.keys())
```

Out[281]:
```
['obj1', 'obj2']
```

In [282]:
```
list(db.values())
```

Out[282]:
```
[[1, 2, 3, 4, 5], (6, 7, 8, 9, 10)]
```

In [283]:
```
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 [284]:
```
db = shelve.open('shl_1')
```

In [285]:
```
db.keys()
```

Out[285]:
```
KeysView(<shelve.DbfilenameShelf object at 0x000002C15AB2BB08>)
```

In [286]:
```
list(db.keys())
```

Out[286]:
```
['obj1', 'obj2']
```

In [287]:
```
list(db.values())
```

Out[287]:
```
[[1, 2, 3, 4, 5], (6, 7, 8, 9, 10)]
```

In [288]:
```
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 размеки
- 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}` , имена (ключи) здесь должны быть строками.
  - Пример: `{"имя1":"строка", "имя2":13, "имя3":true, "имя4":false, "имя5":null}`
- 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 (заключается только в двойные кавычки: `"example"` )
- число (целые и числа с плавающей точкой)
- логическое значение ( `true` , `false` )
- значение `null`

**Создание JSON объектов:**

Импорт библиотек:

In [289]:
```
import json # библиотека для работы с JSON
# import pandas as pd
import requests # библиотека для выполнения HTTP запросов
```

In [249]:
```
my_ab = [] # создаем адресную книгу
```

In [290]:
```
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 [291]:
```
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 [292]:
```
my_ab
```

Out[292]:
```
[{'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'}]},
 {'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 [293]:
```
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 [294]:
```
my_ab2
```

Out[294]:
```
[{'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 [295]:
```
with open('addres-book.json', mode='w', encoding='utf-8') as f: # открываем файл на зап
    json.dump(my_ab, f, indent=2)
```

Как выглядит файл:

In [296]:
```
with open('addres-book.json', 'r', encoding='utf-8') as f:
    for l in f:
        print(l, end="")
```


```
[
  {
    "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"
      }
    ]
  },
  {
    "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 [297]:
```
with open('addres-book.json', 'r', encoding='utf-8') as f: # открываем файл на чтение
    ab_fjs = json.load(f)
```

In [298]:
```
ab_fjs
```

Out[298]:
```
[{'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'}]},
 {'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'}]}]
```

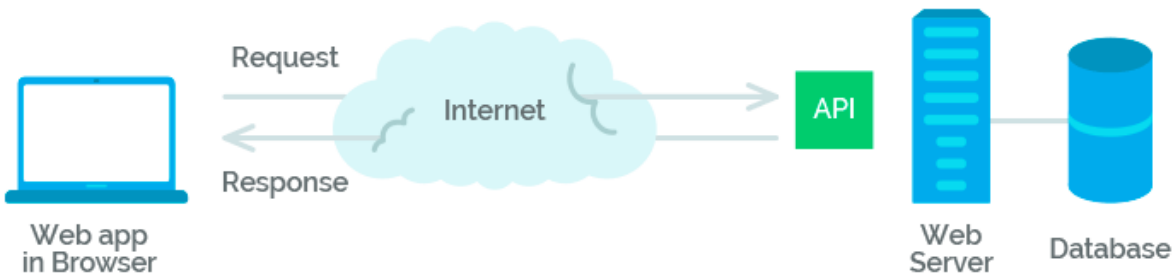
In [299]:
```
ab_fjs[0]['name']
```

Out[299]:
```
'Faina Lee'
```

In [300]:
```
ab_fjs[1]['name']
```

Out[300]:
```
'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)

In [301]:
```
ZIPPOPOTAM = 'http://api.zippopotam.us'
COUNTRY = 'RU'
zip_1 = '125009'
zip_2 = '129337'
```

In [302]:
```
'/'.join((ZIPPOPOTAM, COUNTRY, zip_1))
```

Out[302]:
```
'http://api.zippopotam.us/RU/125009'
```

In [303]:
```
def url_rus_zip(zip_code):
    return '/'.join((ZIPPOPOTAM, COUNTRY, zip_code))
```

In [304]:
```
url_rus_zip(zip_1)
```

Out[304]:
```
'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 [305]:
```
# responce for HTTP GET request from http://api.zippopotam.us
r = requests.get(url_rus_zip(zip_2))
r
```

Out[305]:
```
<Response [200]>
```

In [306]:
```
r.content
```

Out[306]:
```
b'{"post code": "129337", "country": "Russia", "country abbreviation": "RU", "places":
[{"place name": "\\u041c\\u043e\\u0441\\u043a\\u0432\\u0430 337", "longitude": "45.666
7", "state": "\\u041c\\u043e\\u0441\\u043a\\u0432\\u0430", "state abbreviation": "",
"latitude": "60.15"}]}'
```

In [307]:
```
# Convert response into a python object
data = r.json()
```

In [308]:
```
# View the data
data
```

Out[308]:
```
{'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 [309]:
```
# One level deep
data['places']
```

Out[309]:
```
[{'place name': 'Москва 337',
  'longitude': '45.6667',
  'state': 'Москва',
  'state abbreviation': '',
  'latitude': '60.15'}]
```

In [310]:
```
# One level deep, second element
data['places'][0]
```

Out[310]:
```
{'place name': 'Москва 337',
 'longitude': '45.6667',
 'state': 'Москва',
 'state abbreviation': '',
 'latitude': '60.15'}
```

In [311]:
```
# One level down, then second item, then it's longitude object
data['places'][0]['longitude']
```

Out[311]:
```
'45.6667'
```

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

In [312]:
```
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 [313]:
```
# Run the function
extract_latlong(data)
```

```
60.15 45.6667
```

# XML

- к оглавлению

## Формат XML

- [к оглавлению](#)

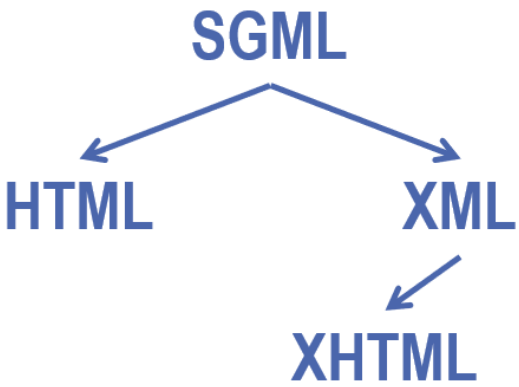
**XML** (англ. eXtensible Markup Language) — расширяемый язык разметки.

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,

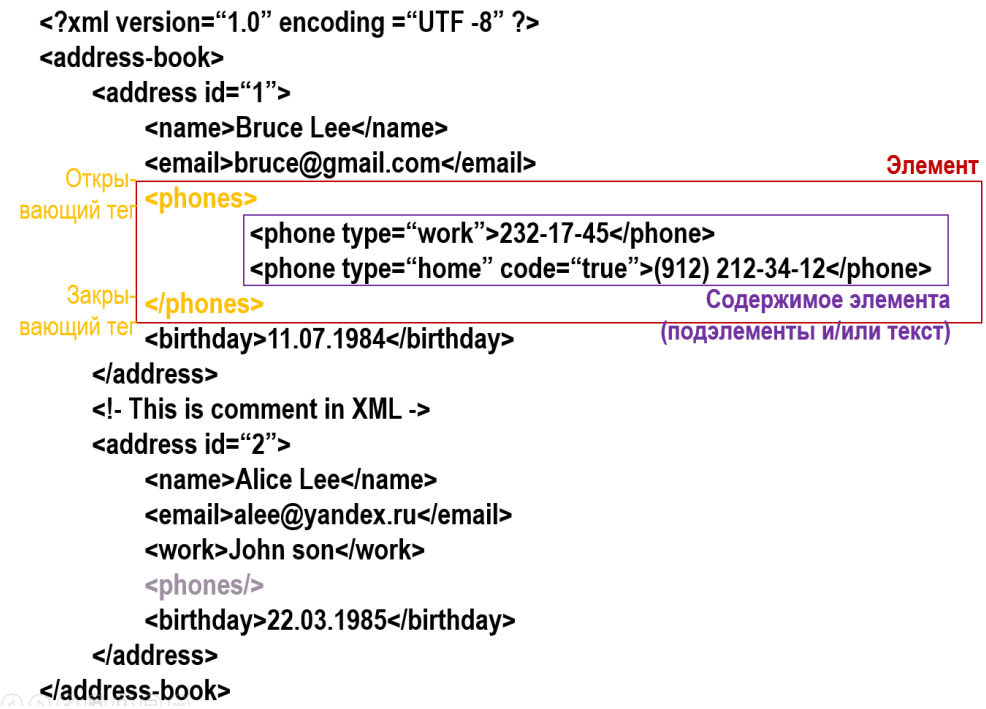
Сравнение XML и HTML:

| HTML | XML |
|---|---|
| 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. |
| 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 version="1.0"?>
<note xmlns="http://www.w3schools.com" xmlns:xsi="http://www.w3.org/2001/XM
LSchema-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://
www.w3schools.com" xmlns="http://www.w3schools.com" elementFormDefault="qualifi
ed">
    <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**


**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:

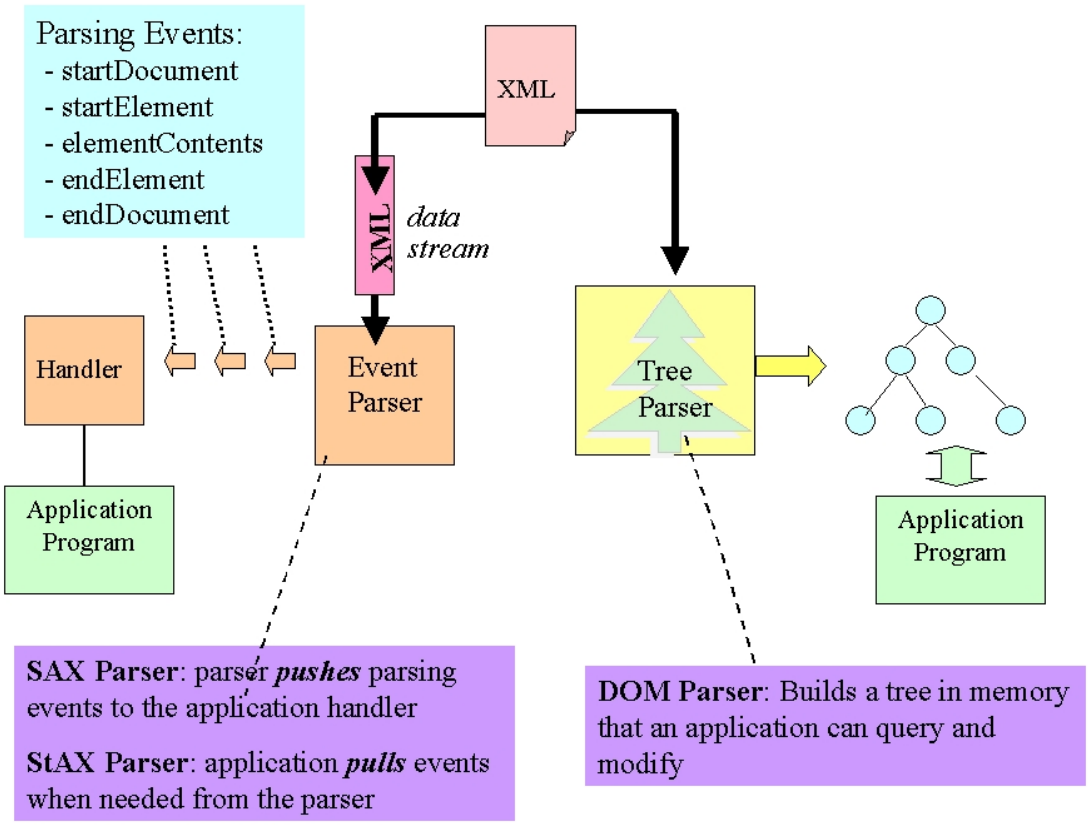
Преимущества JSON:

- **+** 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.

**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.

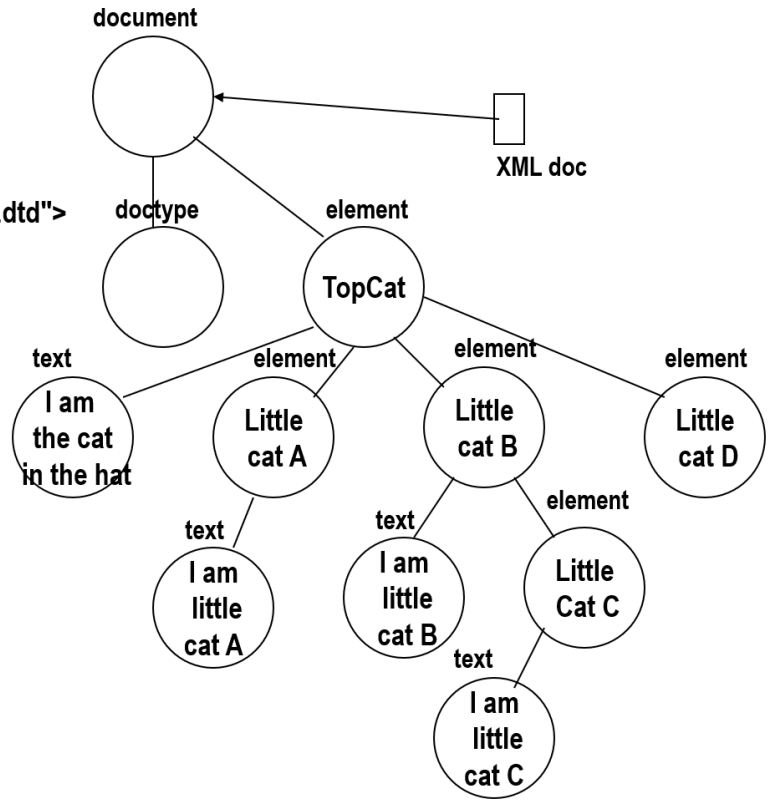
SAX (Simple API for XML)

- 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**

**cats.xml**

```
<?xml version = "1.0" ?>
<!DOCTYPE TopCat SYSTEM "cats.dtd">
<TopCat>
      I am The Cat in The Hat
      <LittleCatA>
          I am Little Cat A
      </LittleCatA>
      <LittleCatB>
           I am Little Cat B
          <LittleCatC>
              I am Little Cat C
          </LittleCatC>
      </LittleCatB>
    <LittleCatD/>
</TopCat>
```




**Пример представления XML документа в виде DOM-дерева**

## Работа с XML в Python

- 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 [466]:
```
# import required modules
import requests
from bs4 import BeautifulSoup
```

Пример для работы:

```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">+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 [472]:
```python
# Выполняем парсинг XML файла:
with open('.\\addres-book.xml') as f:
    ab = BeautifulSoup(f, 'xml')
```

In [473]:
```python
ab
```

Out[473]:
```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 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;amp;son</work>
<phones/>
<birthday>22.03.1985</birthday>
</address>
</address_book>
```

In [474]:
```
# Переходим к корню дерева документа:
ab.address_book
```

Out[474]:
```
<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 [475]:
```
ab.address_book.find_all('address')
```

Out[475]:
```
[<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>,
 <address id="2">
 <name>Alice Lee</name>
 <email>alee@yandex.ru</email>
 <work>John&amp;son</work>
 <phones/>
 <birthday>22.03.1985</birthday>
 </address>]
```

In [476]:
```
ab.address_book.address.name
```

Out[476]:
```
'address'
```

In [477]:
```
ab.address_book.address['id']
```

Out[477]:
```
'1'
```

In [479]:
```
ab.address_book.address.text
```

Out[479]:
```
'\nBruce Lee\nbruce@gmail.com\n\n232-17-45\n+7 (912) 212-34-12\n\n11.07.1984\n'
```

In [480]:
```
# Получаем все дочерние строки разделенные заданным разделителем.
ab.address_book.address.getText('|',strip=True)
```

Out[480]:
```
'Bruce Lee|bruce@gmail.com|232-17-45|+7 (912) 212-34-12|11.07.1984'
```

In [481]:
```
ab.address_book.address.contents
```

Out[481]:
```
['\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 [482]:
```
# 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 [483]:
```
# получаем первый элемент соответствующий указанному пути:
ab.address_book.address.phone
```

Out[483]:
```
<phone type="work">232-17-45</phone>
```

In [484]:
```
# получаем все дочерние элементы 'phone' для пути address_book.address:
ab.address_book.address.find_all('phone')
```

Out[484]:
```
[<phone type="work">232-17-45</phone>,
 <phone code="true" type="home">+7 (912) 212-34-12</phone>]
```

In [485]:
```
# ищем элементы соответствующие более сложному условию:
ab.address_book.address.find('phone', type='home')
```

Out[485]:
```
<phone code="true" type="home">+7 (912) 212-34-12</phone>
```

In [486]:
```
ab.address_book.address.find('phone', type='home')
```

Out[486]:
```
<phone code="true" type="home">+7 (912) 212-34-12</phone>
```

In [487]:
```
ab.address_book.address.find('phone', attrs={'type': 'home'})
```

Out[487]:
```
<phone code="true" type="home">+7 (912) 212-34-12</phone>
```

In [488]:
```
ph1 = ab.address_book.address.find('phone')
```

In [489]:
```
ph1
```

Out[489]:
```
<phone type="work">232-17-45</phone>
```

In [491]:
```
ph1.next_sibling.next_sibling
```

Out[491]:
```
<phone code="true" type="home">+7 (912) 212-34-12</phone>
```

In [492]:
```
list(ph1.next_siblings)
```

Out[492]:
```
['\n', <phone code="true" type="home">+7 (912) 212-34-12</phone>, '\n']
```

In [493]:
```
ph1.next
```

Out[493]:
```
'232-17-45'
```

In [494]:
```
list(ph1.nextGenerator())
```

Out[494]:
```
['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 [495]:
```
ph1.findNextSibling()
```

Out[495]:
```
<phone code="true" type="home">+7 (912) 212-34-12</phone>
```

Более крупный пример:

```
<?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 [496]:
```
with open('.\\addres-book-q.xml') as f:
    ab = BeautifulSoup(f, 'xml')
```

In [497]:
```
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[497]:
```
[{'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>
```

---

# Спасибо за внимание!

# Спасибо за внимание!

---

TODO: requests (краулинг),

- CSV
- ? mime type
- В других лекциях: numpy-native, xlsx, sqlite

In [ ]: