

Zpracování a vizualizace dat v prostředí Python

Získávání dat a datová perzistence

Zdeněk VAŠÍČEK

Fakulta Informačních Technologií, Vysoké Učení Technické v Brně

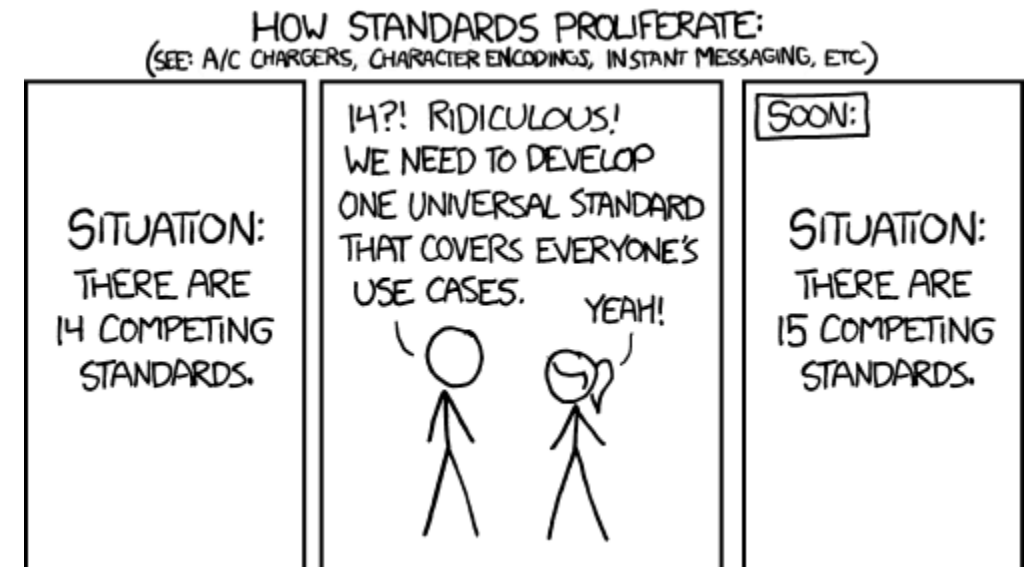
Brno, Czech Republic

vasicek@fit.vutbr.cz



Obsah

- Získávání dat (volně dostupná data, data scraping)
 - nestrukturovaná / špatně strukturovaná data
 - částečně / plně strukturovaná data
- Typické formáty pro výměnu dat a jejich zpracování
 - textové a binární datové formáty
- Serializace a uchovávání dat
 - souborově orientované úložiště
 - databázové systémy



<https://xkcd.com/927>

Členění dat z pohledu jejich struktury

Strojově čitelná data lze s ohledem na automatické počítačové zpracování rozdělit na

■ Strukturovaná data

- Organizovaná data s daným modelem (snadná adresace položek, snadné vyhledávání).
- Lze uchovávat v klasických relačních databázích (SQL DB založených na tabulkách).

■ Částečně strukturovaná data

- Jsou typem strukturovaných dat, ale jednotlivé entity mohou mít svůj vlastní organizační vzor.
- Formát XML a JSON patří mezi typické výměnné formáty této kategorie dat.
- Nepasují do tradičních relačních databází, vhodné spíše pro dokumentově orientované NoSQL databáze uchovávající dvojice klíč-hodnota.

■ Nestrukturovaná data

- Nemají definovaný model nebo organizaci (netriviální vyhledávání).
- Typicky se jedná o data jejichž konzumentem je člověk (tj. multimediální soubory, maily, chat, webový obsah, ale i PDF, kde je důraz na vizuální stránku).
- Nevhodné pro relační DB, spíše dokumentově orientované databáze.



Konkrétní příklady

■ Nestrukturovaná/špatně strukturovaná data

- PDF (v této podobě vyžaduje extrémní úsilí aby bylo možné strojově zpracovat – přesnost dat?)
<https://kod.brno.cz/dataset/.../download/cizinci-v-brne-kniha.pdf> (výsledky výzkumu, link z data.gov.cz)
- HTML layout (změna rozložení = předělání skriptu)
<http://portal.chmi.cz/historicka-data/pocasi/uzemni-teploty>
<https://www.mpsv.cz/prehled-o-vyvoji-castek-minimalni-mzdy>
- Word dokument (automaticky extrémně obtížně zpracovatelné)
<https://www.mpsv.cz/web/cz/zprava-o-zakladnich-tendencich-prijmove-a-vydajove...>

■ Částečně strukturovaná data

- XLSX (potřebujeme další meta informace, abychom správně načetli obsah)
<https://www.mpsv.cz/web/cz/zprava-o-zakladnich-tendencich-prijmove-a-vydajove...>
- CSV
<http://portal.chmi.cz/historicka-data/pocasi/denni-data/Denni-data-dle-z.-123-1998-Sb>
- XML bez schématu
http://portal.chmi.cz/files/portal/docs/uoco/web_generator/AlMdata_hourly.xml

■ Strukturovaná data

- XML vybavený schématem (XSD)
http://wwwinfo.mfcr.cz/cgi-bin/ares/darv_std.cgi?ico=00216305
- JSON vybavený schématem (JSON-schema)
<https://data.gov.cz/datov%C3%A1-sada?iri=...%2F706529437%2F33f8a75fb7dcf5d2314ce667827d1350>

Zdroje dat, další informace

■ Příklady zdrojů dat v ČR

- Portál veřejných dat <https://data.gov.cz/>
- Datové sady ministerstev <https://opendata.mzcr.cz/>, <https://data.mfcr.cz/>, <https://opendata.mzp.cz/>, <https://data.msmt.cz/> (0), <http://data.mmr.cz/>
- Datový portál města Brna <https://data.bрно.cz/>, Prahy <https://opendata.praha.eu/> , ...
- Datové služby KN (adresy, správní jednotky, budovy, ...) <https://services.cuzk.cz/>, <https://atom.cuzk.cz/>
- Mapové služby KN (vektorová data) <https://geoportal.cuzk.cz/>
- Meteorologická data <http://portal.chmi.cz/>
- Polohy vozidel IDS JMK <https://mapa.idsjmk.cz/> , IDS Praha <https://mapa.pid.cz/>

■ Situace s daty v ČR

- viz <https://www.irozhlas.cz/zpravy-tag/datova-zurnalistika>

■ Další informace

- viz <https://stevesie.com/docs/pages/is-data-scraping-legal>

Získávání dat z webových a jiných zdrojů

Přehled možností

Umístění zdroje dat

- Data jsou typicky dostupná pod nějakou URL (Uniform Resource Locator) adresou
 - řetězec ASCII znaků (vyjma mezery) s definovanou strukturou, který slouží k přesné specifikaci umístění zdrojů informací (ve smyslu dokument nebo služba) na Internetu.
- Formát URL:

```
protokol://server.doména_druhého_řádu.generická_doména:port/umístění_na_serveru?query_string#kotva
```

 - příklad: <https://www.httpbin.org/get?param1=value1¶m2=value2%26value3+value4#intro>
- Kódování URL:
 - Znaky mimo ASCII jsou nahrazeny znakem `%` následovaným dvěma hexa číslicemi
 - Mezera je nahrazena znakem `+` nebo pomocí `%20`
- Práce s URL adresou v Python:
 - podpora pro parsování, vytváření a manipulaci s URL je součástí vestavěného modulu `urllib.parse`

Operace nad URL adresou

■ Parsování URL a QS

```
url = up.urlparse('https://user:password@www.httpbin.org:443/get?param1=value1&param2=value2%26value3+value4#intro')
#ParseResult(scheme='https', netloc='user:password@www.httpbin.org:443', path='/get', params='',
query='param1=value1&param2=value2%26value3+value4', fragment='intro')
url.username, url.password, url.hostname, url.port
#('user', 'password', 'www.httpbin.org', 443)
up.parse_qs(url.query)
#{'param1': ['value1'], 'param2': ['value2&value3 value4']}
```

■ Sestavení URL a změna jednotlivých částí

```
newurl = url._replace(netloc='httpbin.org', query = up.urlencode({'param1':'new/val1'}))
parse.urlunparse(newurl) #'https://httpbin.org/get?param1=new%2Fval1#intro'
```

■ Další operace:

- kódování query a path: `urlencode`, `quote`, `quote_plus`, nahrazení path: `urlsplit`, `urljoin`

Získávání dat v prostředí Python

■ Podpora pro získávání dat z webových služeb (HTML, REST, GraphQL, SOAP, ...)

Protokol	Modul	Popis	* viz také high-level framework scrapy a selenium
HTTP/1.1, HTTPS/1.1	<code>urllib.request</code> (vestavěný)	dobrá funkcionality, horší API	
	<code>requests</code> (externí)	propracované API, široká funkcionality, podpora REST	
HTTP/2, HTTPS/2	<code>httpx</code> (externí)	beta verze, asynchronní dotazy, API jako requests	
SOAP, WSDL	<code>zeep</code> (externí)	podpora WSDL, XSD schémat	
gRPC	<code>grpcio</code> (externí)	široká funkcionality	

■ Podpora pro získávání dat z ostatních služeb

Protokol	Modul	Popis
FTP, FTPs	<code>ftplib</code> (vestavěný)	kompletní funkcionality
	<code>requests_ftp</code> (externí)	základní funkcionality, API kompatibilní s requests
SSH, SFTP	<code>paramiko</code> (externí)	široká funkcionality
WebSocket	<code>websockets</code> (externí)	široká funkcionality, asynchronní zpracování

Získávání dat s využitím urllib.request

- Funkce modulu vrací objekt typu HTTPResponse (viz [dokumentace](#))
 - HTTPResponse se chová jako soubor (je iterovatelný, lze jej použít jako context manager)
- Základní použití:
 - metoda GET

```
with urllib.request.urlopen('http://httpbin.org/get?param1=value1', timeout=10) as f:  
    f.read(100).decode('utf-8')  
    for x in iter(f):  
        print(x)
```

- metoda POST

```
with urllib.request.urlopen('http://httpbin.org/post', data=b'post data') as f:  
    print(f.read().decode('utf-8'))
```

- Upřesnění dotazu pomocí třídy Request

```
req = urllib.request.Request(url='http://httpbin.org/headers', method='GET',  
                             headers={'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) ...'})  
with urllib.request.urlopen(req) as f:  
    print(f.read().decode('utf-8'))
```



Získávání dat s využitím urllib.request

■ Autentizace (BasicAuth, DigestAuth)

```
import urllib.request

auth_handler = urllib.request.HTTPBasicAuthHandler()
auth_handler.add_password(realm='Fake Realm', #viz WWW-Authenticate: Basic realm="Fake Realm"
                        uri='httpbin.org',
                        user='user',
                        passwd='p4ssw0rd')

opener = urllib.request.build_opener(auth_handler)
urllib.request.install_opener(opener)

with urllib.request.urlopen('https://httpbin.org/basic-auth/user/p4ssw0rd') as f:
    print(f.read())
```

■ Persistence cookies

```
import os, http.cookiejar, urllib.request
cj = http.cookiejar.MozillaCookieJar()
cj.load(os.path.join(os.path.expanduser("~"), "path", "cookies.txt"))
opener = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(cj))
# urllib.request.install_opener(opener)

with opener.open("https://httpbin.org/cookies/set?cookie1=value1") as f:
    print(f.read().decode('utf-8'))
```



Získávání dat s využitím modulu requests

- Requests nabízí oproti vestavěnému modulu propracovanější API a širší funkcionalitu
- Výčet vlastností z [dokumentace](#):
 - Keep-Alive & Connection Pooling (hlavička [Connection](#))
 - International Domains and URLs
 - Sessions with Cookie Persistence
 - Browser-style SSL Verification (incl. Client side certificates)
 - **Automatic Content Decoding** (hlavička [Content-Type](#); text/html, text/json, ...)
 - Basic/Digest Authentication (hlavička [WWW-Authenticate](#); basic, digest, bearer, ...)
 - Elegant Key/Value Cookies (hlavičky [Set-Cookie](#), [Cookie](#))
 - **Automatic Decompression** (hlavička [Content-Encoding](#); gzip, deflate, br, ...)
 - **Unicode Response Bodies**
 - HTTP(S) Proxy Support
 - **Multipart File Uploads** (hlavička [Content-Type](#): multipart/form-data)
 - **Streaming Downloads**
 - Connection Timeouts
 - **Chunked Requests** (hlavička [Transfer-Encoding](#); chunked)

Získávání dat s využitím requests

■ Příklady použití:

- metoda GET

```
resp = requests.get('https://httpbin.org/get', params={'param1': 'value1'})
resp.status_code
print(resp.text)
```

- metoda POST

```
resp = requests.post('https://httpbin.org/post', data={'param2': 'value2'})
print(resp.text)
```

- hlavičky

```
resp = requests.get('https://httpbin.org/headers', headers={'User-Agent': 'XYZ'})
resp.headers.get('content-type')
print(resp.text)
```

- metody pro konverzi výstupu:

```
jsn = requests.get('https://httpbin.org/get', params={'param1': 'value1'}).json()
#jsn['args']['param1']
```

Autentizace v request

■ Nejběžnější podporované metody autentizace

- HTTP Basic, HTTP Digest, OAuth1, případně OAuth2 prostřednictvím balíku **requests-oauthlib**

■ Příklady:

- HTTP Basic

```
from requests.auth import HTTPBasicAuth
with requests.get('https://httpbin.org/basic-auth/user/p4ssw0rd',
                  auth=HTTPBasicAuth('user', 'p4ssw0rd')) as f:
    print(f.json())
```

- HTTP Basic je výchozí způsob autentizace

```
requests.get('https://httpbin.org/basic-auth/user/p4ssw0rd',
             auth=('user', 'p4ssw0rd'))
    .json()
```

- autentizace pro více volání s využitím requests.Session

```
s = requests.Session()
s.auth = ('user', 'p4ssw0rd')
s.get('https://httpbin.org/basic-auth/user/p4ssw0rd').json()
s.get('https://httpbin.org/get').json()
```



Cookies v requests

■ Persistence cookies

- objekt Session uchovává cookies automaticky (viz atribut s.cookies)

```
with requests.Session() as s:  
    s.get('https://httpbin.org/cookies/set?cookie1=value1')  
    print(s.cookies) #<RequestsCookieJar[<Cookie cookie1=value1 for httpbin.org/>]>  
    print(s.get('https://httpbin.org/cookies').json()) #{'cookies': {'cookie1': 'value1'}}  
    print(requests.get('https://httpbin.org/cookies').json()) #{'cookies': {}}
```

- jako úložiště lze použít výchozí requests.cookies.RequestsCookieJar

```
jar = requests.cookies.RequestsCookieJar()  
jar.set('mycookie', 'value', domain='httpbin.org', path='/cookies')  
with requests.Session() as s:  
    s.cookies = jar  
    print(s.get('https://httpbin.org/cookies').json()) #{'cookies': {'mycookie': 'value'}}  
    print(requests.get('https://httpbin.org/cookies', cookies=jar).json()) #{'cookies': {'mycook
```

- nebo jiného správce, např. http.cookiejar.MozillaCookieJar (viz př. urllib) nebo http.cookiejar.LWPCookieJar

```
with requests.Session() as s:  
    s.cookies = http.cookiejar.LWPCookieJar("cookies.txt")  
    s.get("https://google.com")  
    s.cookies.save()  
    s.cookies.load()
```

Získávání dat s využitím requests

■ Podpora pro práci s objemnými daty

- **upload** (vstupem může být generátor, iterátor, tj. i soubor)

```
with open('file.bin', 'rb') as f:  
    requests.post('http://some.url/streamed', data=f)
```

```
def gendata():  
    yield b'abc'  
    yield b'def'  
requests.post('https://httpbin.org/post', data=gendata()).json()
```

- **download** (stream režim, vhodné na multimediální soubory)

```
with requests.get('https://httpbin.org/stream/100', stream=True) as r:
```

```
    for chunk in r:  
        print(chunk)
```

```
    for chunk in r.iter_content(chunk_size=128, decode_unicode=True):  
        fp.write(chunk)
```

```
    for line in r.iter_lines():  
        print(line)
```


Příklad na využití stream režimu

```
ID = 'CmPA7zE8mx0'
UA = UserAgent().chrome
ROOT = 'https://www.youtube.com'
with requests.Session() as s:
    s.cookies = http.cookiejar.LWPCookieJar("youtube.txt")
    s.headers.update({'User-Agent': UA})

    s.get(f'{ROOT}/watch?v={ID}')

    info = s.get(f'{ROOT}/get_video_info?video_id={ID}&el=embedded&ps=default&eurl=&gl=US&hl=en').text
    info = urllib.parse.parse_qs(info)
    info['player_response'] = json.loads(info['player_response'][0])

    ... # best <= info['player_response']['streamingData']['adaptiveFormats'][X]

    with s.get(best['url'], stream=True) as r:
        with open(f'{ID}-video.mp4', 'wb') as fd:
            for chunk in r.iter_content(chunk_size=16384):
                fd.write(chunk)

    ...
s.cookies.save()
```

Transparentní cache

■ Možnosti realizace transparentní cache

- Jednoduchá perzistentní cache paměť pomocí rozšíření requests_cache (viz [dokumentace](#))

```
import requests, requests_cache
requests_cache.install_cache('cache_filename')
for i in range(10):
    resp = requests.get('https://httpbin.org/delay/1')
    print(resp.from_cache)
```

- Plnohodnotná stavová HTTP cache pomocí rozšíření CacheControl (viz [dokumentace](#))

```
import requests
from cachecontrol import CacheControl
from cachecontrol.caches.file_cache import FileCache

sess = requests.session()
with CacheControl(sess, cache=FileCache('cache_dir')) as cs:
    for i in range(10):
        resp = cs.get('https://www.google.com/tia/tia.png')
        print(resp.from_cache)
```



Příklad na využití dekorátoru (zjednodušený)

```
def retry(tries, debug=False):  
    def wrapper(f):  
        def inner(*args, **kwargs):  
            for t in range(1, tries+1):  
                if t>1: time.sleep(1)  
                try:  
                    if debug: print(f'pokus #{t}')                    return f(*args, **kwargs)  
                except requests.HTTPError as err:  
                    if debug: print(f'')                    if (t == tries):  
                        raise err  
            return inner  
    return wrapper
```

```
@retry(tries=3, debug=True)  
def download():  
    with requests.get('http://httpbin.org/status/404,403,200', timeout=10) as f:  
        f.raise_for_status()  
        print(f.status_code, f.raw.read())
```

```
download()
```



Formáty dat a jejich zpracování

HTML, XHTML, JSON, XML, CSV, XLSX, XLS, ...

Formáty pro výměnu dat

■ Podpora pro manipulaci s textovými formáty dat

Formát	Modul	Popis
CSV, TSV	<code>csv</code> (vestavěný)	načítání a ukládání dat z/do CSV
HTML/XHTML	<code>html.parser</code> (vestavěný)	čistě parser generující události
HTML/XHTML/XML	<code>beautifulsoup4</code> (externí)	pokročilé API, práce s nevalidním vstupem
XML	<code>xml.etree.ElementTree</code> (vestavěný)	načítání, manipulace, ukládání XML
	<code>lxml</code> (externí)	alternativa pro manipulaci s velkými soubory
JSON	<code>json</code> (vestavěný)	načítání a ukládání dat
XLSX	<code>openpyxl</code> (externí)	načítání, manipulace, ukládání Excel 2010+

■ Binární formáty dat

Formát	Modul	Popis
XLS	<code>xlrd, xlwr</code> (externí)	načítání, manipulace, ukládání Excel 2007
	<code>pyexcel</code> (externí)	univerzální framework pro CSV,XLS,XLSX
PDF	<code>PyPDF2</code> (externí)	načítání, manipulace, ukládání
další	<code>struct</code> (vestavěný), <code>protobuf</code> , ...	

Zpracování CSV

■ Typická sekvence při zpracování dat s využitím vestavěného modulu [csv](#)

- otevření souboru

```
import csv
with open('dotaznik.csv', encoding="utf-8") as csvfile:
```

- vytvoření readeru; automatická detekce nastavení parametrů (nemusí vždy fungovat), nebo

```
dialect = csv.Sniffer().sniff(csvfile.read(1024))
csvfile.seek(0)
reader = csv.reader(csvfile, dialect)
```

- vytvoření readeru; manuální konfigurace

```
reader = csv.reader(csvfile, delimiter=',', quotechar='"')
```

- zpracování vstupu do slovníku

```
header = next(reader) #hlavicka
for row in reader:
    row = dict(zip(header,row))
    ...
```

Zpracování HTML a XML

- Modul [beautifulsoup4](#) umožňuje parsovat, procházet a modifikovat HTML/XML strom

```
from bs4 import BeautifulSoup      #%pip install beautifulsoup4
soup = BeautifulSoup("<html><body>a <b style=xyz>web</b> page<b class=a> 2</b></html>", 'html.parser')
```

- Základní operace:

- pohyb po stromu

```
assert(soup.body == soup.find('body'))
assert(soup.b == soup.find('b'))
assert(soup.b.parent == soup.body)
assert(soup.b.next_sibling == ' page')
assert(soup.b.previous_sibling == 'a ')
```

- vyhledávání

```
soup.find('b', class_='a')
soup.find_all('b', style=lambda val: val=='xyz')
soup.find_all(string=re.compile("e|g")) # podle obsahu ['web', ' page']
soup.select(".a") #CSS selektory
```

- modifikace stromu viz [dokumentace](#)

- přístup k atributům

```
assert(soup.b['style'] == 'xyz')
```

- přístup k obsahu

```
assert(soup.body.text == 'a web page 2')
```

- generování výstupu viz [dokumentace](#)



Serializace dat a datová persistence

Souborová úložiště, databázové systémy

Možnosti serializace dat

- **Data malého a středního objemu, heterogenní data**
 - Formát PICKLE pokud potřebujeme serializovat Python objekty
 - Formát JSON pokud potřebujeme přenositelnost mezi jazyky (extrakce dat z JSON lze např. pomocí [jmespath](#), [pyjsonq](#), dotazování např. pomocí [pythonql3](#))
- **Data velkého objemu, převážně numerická**
 - hierarchické datové úložiště HDF5 (uniformní pole), zvážit nativně podporovanou kompresi
- **Databázové systémy, pokud potřebujeme provádět dotazy**
 - lokální souborově orientované DB
 - SQLite3 – vestavěný modul [sqlite3](#)
 - serverově orientované relační DB
 - PostgreSQL – externí modul [psycopg](#),
 - MySQL – externí modul [mysql-client](#), [mysql-connector-python](#), ...
 - nebo všechny výše uvedené pomocí vysokoúrovňového API (např. [sqlobject](#), [sqlalchemy](#), ...)
 - serverově orientované NoSQL DB
 - MongoDB – externí modul [pymongo](#)

Serializace dat

```
import pickle, datetime, json
```

- Pomocí vestavěného modulu [pickle](#) (jakékoliv Python objekty)

```
data = {'num':123,  
        10: [1, 2, 3, "x", None,  
             datetime.datetime.now()]}
```

- serializace do řetězce

```
serialized_data = pickle.dumps(data)  
data = pickle.loads(serialized_data)
```

- serializace do souboru

```
with open('data.pkl','wb') as f:  
    pickle.dump(data, f)  
    pickle.dump(data2, f)
```

```
with open('data.pkl','rb') as f:  
    data = pickle.load(f)  
    data2 = pickle.load(f)
```

- Pomocí vestavěného modulu [json](#) (pouze serializovatelná data)

```
data = {'num':123,  
        10:[1, 2, 3, "x", None,  
            int(datetime.datetime  
                .now().timestamp())]}
```

- serializace do řetězce

```
serialized_data=json.dumps(data)  
json.loads(serialized_data)
```

- serializace do souboru

```
with open('data.json','wt') as f:  
    json.dump(data, f)  
with open('data.json','rb') as f:  
    data = json.load(f)
```

❓ proč nefunguje po načtení přístup k data[10]?

Redukce velikosti dat pomocí bezeztrátové komprese

■ Podpora kompresních formátů

Formát	Modul	Popis
GZIP	gzip (vestavěný)	základní kompresní metoda využívající algoritmus DEFLATE (LZ77+Huffman)
BZIP2	bz2 (vestavěný)	vyšší kompresní poměr oproti DEFLATE díky BWT (100-900kB), ale pomalejší
XZ	lzma (vestavěný)	lepší kompresní poměr než BZIP2, varianta LZ7 + range encoding, 4GB slovník
LZO	python-lzo (externí)	účinnost podobná DEFLATE, avšak mnohem rychlejší komprese i dekomprese
zlib	zlib (vestavěný)	Implementace DEFLATE bez kontrolních mechanismů

■ Podpora archivace souborů

Formát	Modul	Popis
ZIP	zipfile (vestavěný)	archivace souborů pomocí metod STORE, DEFLATE a případně BZIP2, LZMA
TAR	tarfile (vestavěný)	spojení souborů, komprese probíhá až následně (tar.gz, tar.bz2, tar.xz)
RAR	rarfile (externí)	není nativní podpora, založeno na volá externího aplikace (není open source)

* modul **patool** nabízí univerzální interface na různé externí archivační utility

Využití komprese pro redukci velikosti serializovaných dat

- Serializovaná data (JSON, Pickle, XML, ...) je vhodné komprimovat

```
import json, gzip, bz2, lzma, datetime
data = {'num':123, 10:[1,2,3,"x",None,int(datetime.datetime.now().timestamp())]}
```

- komprese dat při ukládání

```
with gzip.open('data.jsn.gz','wt') as f:
    json.dump(data, f, indent=4)
with bz2.open('data.jsn.bz2','wt') as f:
    json.dump(data, f, indent=4)
with lzma.open('data.jsn.xz','wt') as f:
    json.dump(data, f, indent=4)
```

- načítání komprimovaných dat

```
with gzip.open('data.jsn.gz','rt') as f:
    data = json.load(f)
with bz2.open('data.jsn.bz2','rt') as f:
    data = json.load(f)
json.load(gzip.open('data.jsn.gz','rt'))
json.load(bz2.open('data.jsn.bz2','rt'))
json.load(lzma.open('data.jsn.xz','rt'))
```

Princip práce se ZIP archivem

■ Vytvoření archivu

```
with zipfile.ZipFile('archiv.zip', 'w') as zf:
    for fn in range(0, 10):
        zi = zipfile.ZipInfo(f'file{fn:05d}.txt')
        zi.compress_type=zipfile.ZIP_DEFLATED #nebo ZIP_STORED, ZIP_BZIP2, ZIP_LZMA
        with zf.open(zi, mode='w') as dest:
            dest.write(bytes(f'obsah souboru {zi.filename}\r\n', encoding='ascii'))
            dest.write(b'A'*1_000_000)
```

■ Přístup k souborům v archivu

```
with zipfile.ZipFile('archiv.zip', 'r') as zf:
    print(zf.namelist()) #seznam souborů v archivu
    with zf.open('file00000.txt', 'r') as f:
        print(f.readline()) #b'obsah souboru file00000.txt\r\n'
```

■ Přidání souboru do archivu

```
with zipfile.ZipFile('archiv.zip', 'a') as zf:
    with zf.open('newfile.txt', mode='w') as dest:
        dest.write(b'X'*1_000_000)
```



Porovnání variant archivačních balíčků

- Benchmark: 100 souborů o velikosti 10 MB, 16 kB bloky stejných / náhodných symbolů, SSD disk
- Operace: vytvoření archivu (přímé z generátoru, žádné mezisoubory)

archiv	komprese	uniformní data		náhodná data	
		dobu vytváření	velikost archivu	dobu vytváření	velikost archivu
tar	-	1.6	1 000 099 840	5.3	1 000 099 840
	GZIP	11.4	1 093 813	36.4	1 000 322 739
	BZIP2	11.2	25 299	172.0	1 004 425 810
	LZMA	18.6	151 764	409.2	1 000 138 832
zip	STORED	2.4	1 000 011 522	6.0	1 000 011 522
	DEFLATED	13.1	1 095 830	34.9	1 000 317 022
	BZIP2	13.0	93 032	150.6	1 004 460 785
	LZMA	24.9	180 102	322.4	1 013 606 521

Porovnání variant archivačních balíčků

- Benchmark: 100 souborů o velikosti 10 MB, 16 kB bloky stejných / náhodných symbolů, SSD disk
- Operace: čtení dat z archivu (přímé, žádné mezisoubory)

archiv	komprese	uniformní data			náhodná data		
		list files	read first	read last	list files	read first	read last
tar	-	0.005	0.011	0.017	0.007	0.014	0.012
	GZIP	4.328	5.330	5.242	2.042	3.095	3.191
	BZIP2	3.013	3.397	3.454	96.549	89.709	98.493
	LZMA	1.485	1.895	1.690	2.109	2.758	2.850
zip	STORED	0.003	0.022	0.081	0.003	0.023	0.018
	DEFLATED	0.003	0.059	0.063	0.003	0.031	0.026
	BZIP2	0.003	0.072	0.071	0.004	0.990	0.876
	LZMA	0.003	0.058	0.054	0.002	0.626	0.657

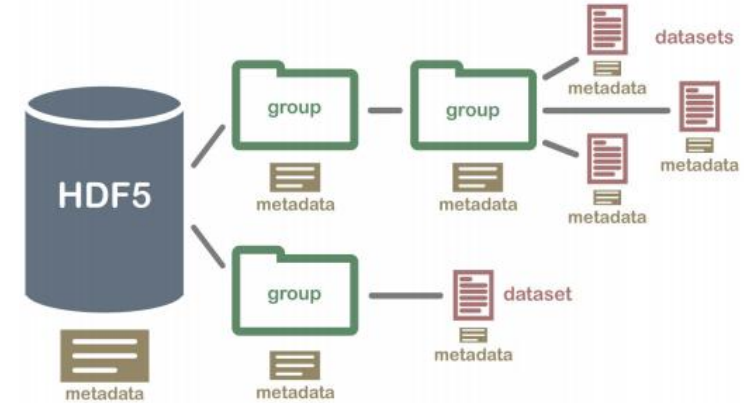
Porovnání variant archivačních balíčků

- **Vlastnosti archivačních formátů ZIP a TAR**
 - Oba formáty umožňují archivaci více souborů
 - Oba formáty umožňují rychlé přidání nových dat na konec souboru
 - ZIP umožňuje rychlý přístup ke kterémukoliv souboru kdykoliv, TAR pouze pokud není aplikována dodatečná komprese (penalizace velikosti několik řádů!! 3 ms vs 96 s)
- **Manipulace s archivy v Python**
 - Práce s TAR není příjemná: neexistuje API pro přímý přístup do TAR. Je zapotřebí přistupovat přes objekt `fileobj` s tím, že je nutné znát strukturu TAR (padding) a velikost dat předem.
 - Podpora komprese u TAR u objektově orientovaného rozhraní je krkolomná
- **Pozor na metodu `seek` u archivu ZIP a TAR**

HDF5 (Hierarchical Data Format)

- Přenosný hierarchický datový formát, navržený pro uchovávání a organizaci velkého množství numerických dat.

- HDF5 = virtuální souborový systém v jednom souboru
- Umožňuje vytvářet v rámci souboru skupiny (groups) datových bodů (dataset), které kromě dat obsahují veškeré metainformace potřebné k interpretaci dat (self-describing format).
- Rychlý přístup k datům (podpora pro extrahování částí číselných řad)
- Přímá podpora v NumPy



- Organizace dat

- datové body lze organizovat hierarchicky do skupin
- datové body určitého datového typu tvoří tzv. dataset – viz (shape, size, dtype) v NumPy array

- Dataset

- dataset může být uložen v souboru kontinuálně nebo po souvislých blocích (tzv. chunk režim, který umožňuje kompresi) přičemž v druhém případě se přístup udržuje přes B-tree
- velikost datasetu může růst pokud je tak specifikováno při jeho vytváření
- řetězce a nenumerická data sice lze uložit, ale ... viz [dokumentace](#)

Manipulace s HDF5

- Ukázka práce s HDF5 pomocí externího modulu h5py (více viz [dokumentace](#))
 - pořízení dat

```
with h5py.File('mytestfile.hdf5', 'w') as f:
    grp = f.create_group("subgroup")

    dset = grp.create_dataset("mydataset", (10_000_000,), dtype='int64')
    dset[:] = 12345

    dset = grp.create_dataset("mydataset_chunked", (10_000_000,), dtype='int64,,
                                chunks=(10_000,))
    dset[1000:] = 12345

    dset = grp.create_dataset("mydataset_chunked2", (10_000_000,), dtype='int64',
                                chunks=(10_000,), compression="gzip", compression_opts=9)
    dset[-1000:] = 12345
```

- přístup k datům

```
f = h5py.File('mytestfile3.hdf5', 'r')
print(f['subgroup']['mydataset_chunked2'][9_000_000])
f.close()
```



Uchování dat v relační DB pomocí sqlalchemy

viz <https://docs.sqlalchemy.org/en/13/orm/tutorial.html>

```
from sqlalchemy import Column, Integer, String, create_engine
from sqlalchemy.ext.declarative import declarative_base
engine = create_engine('sqlite:///memory:', echo=True) #'sqlite:///data.db'
```

```
MyDB = declarative_base()
```

```
class User(MyDB):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    fullname = Column(String)
    password = Column(String)
    def __repr__(self):
        return f"<User(name='{self.name}', fullname='{self.fullname}', password='{self.password}')>"
```

```
MyDB.metadata.create_all(engine)
```

```
session = sessionmaker(bind=engine)()
ed_user = User(name='ed', fullname='Ed Jones', password='edspassword')
session.add(ed_user)
session.commit()
user_by_email = session.query(User).filter(User.name=='ed').first()
print(user_by_email)
```