

Робота з internet-файлами

Доступ до веб-сайтів (модуль `http.client`)

Бібліотека Python забезпечує підтримку протоколу HTTP на стороні клієнта – стандарту структури повідомлень і портів, які використовують для передачі даних в World Wide Web. Це є той самий протокол, який використовує веб-браузер (Internet Explorer чи інший) для отримання веб-сторінок і запуску додатків на віддалених серверах. Фактично, він просто визначає порядок обміну байтами через порт 80.

Стандартний модуль Python `http.client` в значній мірі автоматизує використання протоколу HTTP (або HTTPS) і дозволяє сценаріям отримувати веб-сторінки майже так само, як в веб-браузерах. Вибір протоколу HTTP чи HTTPS важливий, він залежить від сервера.

Схему отримання файла від сервера за допомогою `http.client` можна визначити так:

```
import http.client
server = http.client.HTTPSConnection(servername) # створити об'єкт сервера
server.request('GET', filename)                 # відправити запит
# можна також відправити запит POST, як і імена файлів сценаріїв CGI
reply = server.getresponse()                     # отримати відповідь сервера
if reply.status == 200:                          # код 200 означає успішне виконання
    with open(fname, "wb") as fsave:             # зберегти у файл як потік байтів
        fsave.write(reply.read())               # для "wb"
    reply.close()                                # закрити з'єднання з сервером
else:
    print('Error sending request', reply.status, reply.reason)
```

За такою схемою можна повторно посилати запити `request()`, аналізуючи отриманий файл, і читати наступні відповіді `getresponse()`. На практиці, якщо потрібно лише отримати від сервера цілий файл, використовують простіші схеми.

Програмування сценаріїв веб-клієнтів

Найперше, що потрібно, - знати повну url-адресу сервера і веб-сторінки чи файла, розташованої на сервері. Для цього можна, наприклад, використати звичайний Internet Explorer, виконавши пошук в Internet до отримання кінцевої сторінки веб-сайту (ендпоінт). Internet Explorer показує повну url-адресу в рядку зверху вікна відображення сторінки. Цю адресу треба копіювати у свій сценарій. Для веб-серверів загального доступу такі адреси зазвичай є постійними.

Тепер цю адресу треба поділити на окремі частини: адреса сервера, адреса файла, параметри файла, та інше. Це можна зробити “вручну”, копіюючи частини тексту.

Наприклад, маємо адресу однієї з веб-сторінок Національного банку України з офіційним курсом гривні щодо іноземних валют:

`https://bank.gov.ua/markets/exchangerates/?date=21.11.2019&period=daily`

Адреса сайту: `"bank.gov.ua"`

Адреса файла: `"/markets/exchangerates/?date=21.11.2019&period=daily"`



дата !

Проте, краще використати функції опрацювання url-адрес, які виконують поділ повної url-адреси на частини, або обернену операцію – будову повної url-адреси з окремих частин:

```
import http.client, urllib.parse
anyurl="https://bank.gov.ua/markets/exchangerates/?date=21.11.2019&period=daily"
p = urllib.parse.urlparse(anyurl) # поділити адресу на частини
print(p)
print(p[1])
print(p[2]+p[3]+p[4]+p[5])
```

```
ParseResult(scheme='https', netloc='bank.gov.ua',
path='/markets/exchangerates/', params='', query='date=21.11.2019&period=daily',
fragment='')
bank.gov.ua
/markets/exchangerates/date=21.11.2019&period=daily
```

Отримали кортеж з 6 елементів. Тепер адреса сайту – це `p[1]`, а адреса файла разом з параметрами і уточненнями варіанта – `p[2]+p[3]+p[4]+p[5]`.

Обернена операція:

```
newurl = urllib.parse.urlunparse(p) # будувати url з 6-елементного кортежу
print(newurl)
```

```
https://bank.gov.ua/markets/exchangerates/?date=21.11.2019&period=daily
```

Друге, що потрібно для правильної будови сценарію, - передбачити можливі помилки виконання сценарію за рахунок зовнішніх факторів. Наприклад, може бути неправильним ім'я сервера, ім'я файла на сервері, проблеми зі з'єднанням. Фрагмент сценарію можна будувати так:

```
try:
    server = http.client.HTTPSConnection(servername) # створити об'єкт сервера
    server.request('GET', filename) # відправити запит
    reply = server.getresponse() # отримати відповідь сервера
except: # можуть бути помилки з'єднання
    info = sys.exc_info() # отримати інф. про помилку (type, value, traceback)
    print("Error HTTPSConnection:\n", info[0], info[1])
    sys.exit() # припинити виконання сценарію
```

Третє, що потрібно, це можливість стеження за виконанням сценарію. Оскільки виконується копіювання файлів, то сам сценарій не має якихось друкованих результатів про хід виконання. Зазвичай в таких випадках додають до сценарію оператори трасування: `print("Start...")`, `print("...reply...")`, `print("...ready")` тощо.

Приклад сценарію веб-клієнта (модуль `http.client`)

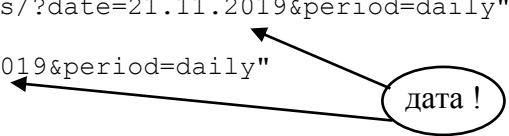
```
# веб-клієнт - прочитати веб-сторінку з веб-сервера
import http.client, urllib.parse, sys
print("Start . . .")

anyurl="https://bank.gov.ua/markets/exchangerates/?date=21.11.2019&period=daily"
servername = "bank.gov.ua"
filename = "/markets/exchangerates/?date=20.11.2019&period=daily"

try:
    server = http.client.HTTPSConnection(servername) # створити об'єкт сервера
    server.request('GET', filename) # відправити запит
    reply = server.getresponse() # отримати відповідь сервера
except: # можуть бути помилки з'єднання
    info= sys.exc_info() #отримати інформацію про помилку (type,value,traceback)
    print("Error HTTPSConnection:\n", info[0], info[1])
    sys.exit() # припинити виконання сценарію

print(". . . reply . . .") # відповідь отримана - продовжуємо
if reply.status == 200: # код 200 означає успішне виконання
    with open("bank-exc.html","wb") as fsave : # зберегти у файл як потік байтів
        fsave.write(reply.read()) # для "wb"
    reply.close() # закрити з'єднання з сервером
else:
    print('Error sending request: ', reply.status, " ", reply.reason)

print(". . . ready")
# можна запустити файл на перегляд одним з способів, розглянутих раніше:
import os
os.startfile("bank-exc.html")
pass # кінець сценарію
```



Тип прочитаного файла можна вказати відразу (`open("bank-exc.html","wb")`), або пізніше виправити у вікні провідника Windows чи іншої службової програми. Тип треба знати наперед відповідно до задачі, яку ми розв'язуємо.

Доступ до файлів на веб-серверах (модуль `urllib.request`)

Якщо потрібно лише отримати від сервера цілий файл, використовують простіші схеми, використовуючи модуль `urllib.request`.

Варіанти схем подано далі.

```
# використовуємо модуль urllib.request
import urllib.request

print("Start . . .")

remoteaddr = "https://bt.rozetka.com.ua/ua/refrigerators/c80125/"
            # файл html
#remoteaddr = "https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange"
            # файл XML
#remoteaddr = "http://wz.lviv.ua/rss" # файл XML

# зберігати треба в файл такого ж типу, звідки отримали
""" # варіант 1: використати функцію urlopen()
remotefile = urllib.request.urlopen(remoteaddr) # об'єкт файла для читання
with open("rozetka-refr.html","wb") as fsave: # зберегти у двійковий файл ("wb")
    fsave.write(remotefile.read()) # прочитати дані напрямо
remotefile.close()
"""

""" # варіант 2: використати функцію urlretrieve()
# копіювати файл або сценарій
local_filename, headers = urllib.request.urlretrieve(remoteaddr)
xml = open(local_filename, "rb") # відкрити локальний файл (двійковий)
with open("bank-esc.xml", "wb") as fsave : # зберегти у файл ("wb")
    fsave.write(xml.read()) # прочитати дані напрямо
xml.close()
"""

print(". . . ready")
# можна запустити файл на перегляд одним з способів, розглянутих раніше:
import os
os.startfile("rozetka-refr.html")
#os.startfile("bank-esc.xml")
pass # кінець сценарію
```

Задача програмування сценаріїв веб-служб

Мовою Python можна створювати веб-служби. Веб-служба передбачає отримання певних даних чи файлів з Інтернету, опрацювання даних (пошук у файлі, фільтрування, перетворення, форматування), відображення опрацьованих даних.

Отримання даних виконують за схемами, викладеними вище.

Опрацювання даних повністю залежить від від їх типу і структури отриманого з Інтернету файла. Це питання вирішують окремо для типів файлів і типів задач.

Відображення опрацьованих даних можна виконати за схемами, викладеними в темі “Деякі загальні методи організації сценаріїв”.

Веб-службу будують окремим сценарієм, який можна запускати, наприклад, натисканням на піктограму на робочому столі (піктограму треба створити), або автоматично, записавши ім'я сценарію до переліку *планувальника завдань* Windows і визначивши розклад до виконання.

Опрацювання xml-файлів

XML-файли будують за допомогою тегів, які забезпечують структуру змістового наповнення даних файла. Структура xml-файла на рівні тегів є деревом, яке має вузли, підлеглі вузли, атрибути вузлів, а також інші елементи.

Для опрацювання xml-файла його читають, ділять на текстові лексеми і будують дерево вузлів (parse). Кожен вузол має функції і класи, визначені в модулі xml.etree.ElementTree, які забезпечують необхідне опрацювання вузлів і цілого документа.

Точне визначення функцій треба шукати в довідковій системі Python. Крім того, треба звертатись за документацією цілого пакета модулів xml.

Опрацювання xml-файла фактично зводиться до пересування деревом вузлів і читання даних у потрібних вузлах.

Текст xml-файла "data.xml": (приклад)

```
<?xml version="1.0"?>
<info>
<magazine title="Магазин 'Ваші меблі'" address="вул.Південна,28" >
  <timetable work="Працюємо щоденно від 9 до 20 години без вихідних" />
</magazine>
<data list="Меблі для квартир та офісів" >
  <furniture name="Стіл">
    <rank>Серія: меблі квартирні </rank>
    <production>Виробництво: Стрийська меблева фабрика</production>
    <price>ціна 5670 грн.</price>
    <service sale="Оплата: перерахунок" delivery="Доставка: магазин,власна"/>
    <service mounting="монтаж: майстер,власний" guarantee="гарантія: 4 роки"/>
  </furniture>
  <furniture name="Стілець">
    <rank>Серія: меблі офісні</rank>
    <production>Виробництво: фабрика 'Закарпаття'</production>
    <price>ціна 950 грн.</price>
    <service sale="Оплата: перерахунок,готівка" delivery="Доставка: власна"/>
  </furniture>
  <furniture name="Диван">
    <rank>Серія: меблі квартирні</rank>
    <production>Виробництво: компанія 'Меблі-сервіс'</production>
    <price>ціна 8400 грн.</price>
    <service sale="Оплата: перерахунок" delivery="Доставка: магазин,власна"/>
    <service mounting="монтаж: майстер,власний" guarantee="гарантія: 3 роки"/>
  </furniture>
</data>
</info>
```

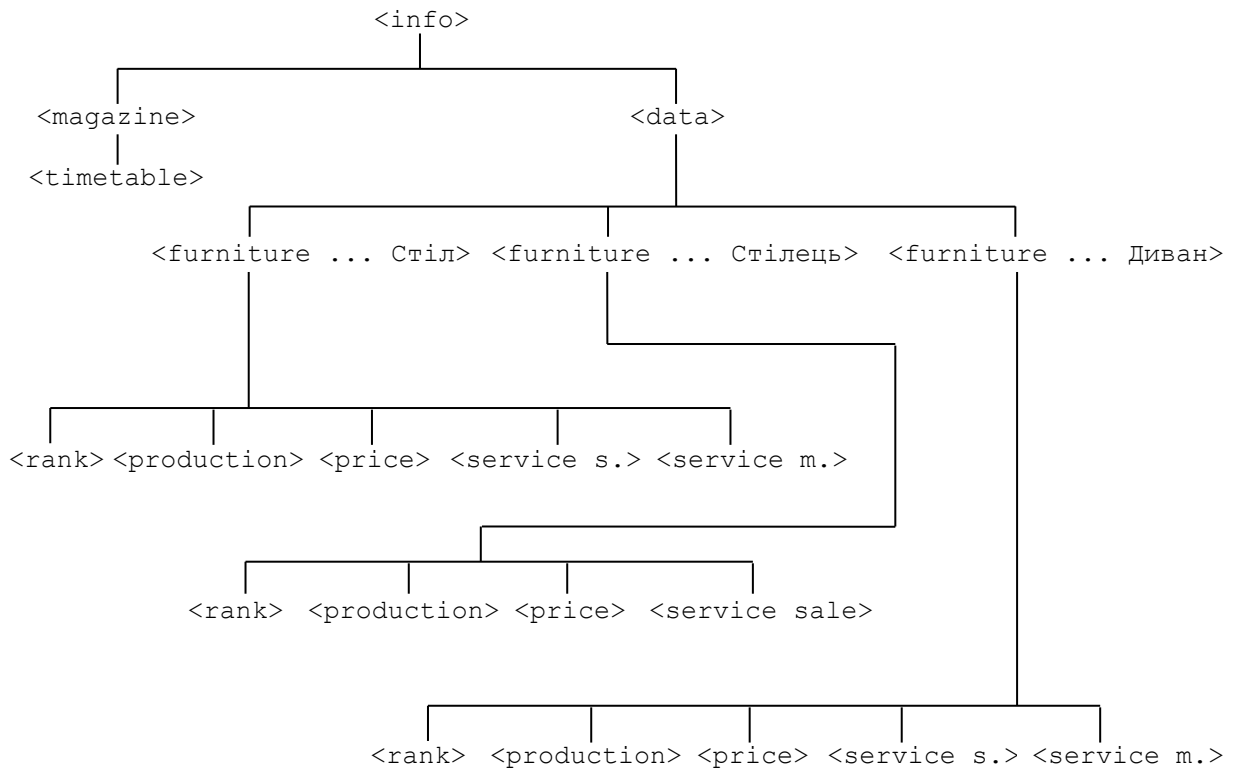
Для такого файла дерево вузлів буде виглядати так, як далі на рисунку.

Подаємо приклад опрацювання цього xml-файла з демонстрацією окремих функцій модуля xml.etree.ElementTree. В прикладі отримані з xml-файла дані просто друкують, а на практиці їх записують в потрібні об'єкти сценарію, форматують до потрібного вигляду і відображають на екрані чи записують в файл.. Надруковані значення обведені рамкою.

```
# опрацювання xml-файлів: використати модуль xml.etree.ElementTree
import xml.etree.ElementTree as ET # ET - скорочене позначення

# читати дані з xml-файла: текст файла має бути в коді UTF-8
tree = ET.parse('data.xml') # читати і розділити на окремі елементи

# кожний елемент дерева має tag і словник атрибутів attrib
root = tree.getroot() # кореневий елемент (вузол, атрибут)
print(root.tag, root.attrib)
print()
info {}
```



```

# дочірні вузли кожного атрибута є ітерованим об'єктом і мають свої атрибути:
for child in root:
    print(child.tag, child.attrib)
print()

```

```

magazine {'title': "Магазин 'Ваші меблі'", 'address': 'вул.Південна,28'}
data {'list': 'Меблі для квартир та офісів'}

```

```

"""

```

```

Не всі елементи XML будуть опрацьовані в дереві parse.
Модуль xml.etree.ElementTree пропускає XML-коментарі, директиви опрацювання
файла, і визначення типів в документі.
"""

```

```

# доступ до дочірніх вузлів виконують додатковими індексами,
# а доступ до словника атрибутів - як у звичайного словника - через ключі
print(root[1].attrib["list"], root[1][0].attrib["name"], root[1][0][2].text,
sep='\n')
print()

```

```

Меблі для квартир та офісів
Стіл
ціна 5670 грн.

```

```

# пошук потрібних елементів:
# кожний вузол має метод iter(), який будує ітератор для обходу
# всіх вузлів заданого імені цілого піддерева даного вузла
for serv in root.iter('service'):
    print(serv.attrib)
print()

```

```

{'sale': 'Оплата: перерахунок', 'delivery': 'Доставка: магазин, власна'}
{'mounting': 'монтаж: майстер, власний', 'guarantee': 'гарантія: 4 роки'}
{'sale': 'Оплата: перерахунок, готівка', 'delivery': 'Доставка: власна'}
{'sale': 'Оплата: перерахунок', 'delivery': 'Доставка: магазин, власна'}
{'mounting': 'монтаж: майстер, власний', 'guarantee': 'гарантія: 3 роки'}

```

```
# інший спосіб пошуку:
for furn in root.iter('furniture'): # в цілому дереві обійти всі вузли furniture
    print(furn.attrib["name"], ":")
    for serv in furn.findall('service'): # для furn знайти підвузли service
        # для кожного підвузла отримати доступ до його атрибутів
        sa = serv.get("sale")
        de = serv.get("delivery")
        mn = serv.get("mounting")
        gu = serv.get("guarantee")
        if sa : print(sa, " ", end="")
        if de : print(de)
        if mn : print(mn, " ", end="")
        if gu : print(gu)
    print()
```

```
Стіл :
Оплата: перерахунок    Доставка: магазин, власна
монтаж: майстер, власний    гарантія: 4 роки
```

```
Стілець :
Оплата: перерахунок, готівка    Доставка: власна
```

```
Диван :
Оплата: перерахунок    Доставка: магазин, власна
монтаж: майстер, власний    гарантія: 3 роки
```

```
# ----- кінець сценарію -----
```

Зауважимо, що для xml-файла можна не лише вести пошук даних, але й редагувати файл, а також будувати новий xml-файл. Документація пакету модулів xml і, зокрема, модуля xml.etree.ElementTree має опис переліку засобів редагування і створення xml-файлів.