

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA



BÁO CÁO TỔNG KẾT MỞ RỘNG
MÔN HỌC XỬ LÝ SỐ TÍN HIỆU

Đề tài

Chuyển đổi từ EXCEL sang định dạng XML
và ngược lại theo quy chuẩn của bộ y tế.

Giáo viên hướng dẫn: Thạc sĩ Nguyễn Thanh Tuấn

Sinh viên thực hiện:

Thái Quang Nguyên 1813294

TP Hồ Chí Minh, Ngày 12/12/2020

Mục Lục

I.	<i>Dẫn nhập về định dạng XML:</i>	3
1.	XML là gì? Tác dụng của file XML:	3
2.	Đặc điểm của XML:	3
II.	<i>Cấu tạo thư mục & thư viện cần có:</i>	4
1.	Cấu tạo thư mục:	4
2.	Các thư viện cần có:	6
III.	<i>Diễn giải code:</i>	6
1.	Chuyển từ EXCEL về XML:	6
a.	Cách thức đọc file EXCEL:	6
b.	Tạo class XMLNode:	8
c.	Scan file EXCEL:	9
d.	Đọc và lưu xml các trường dữ liệu bằng đệ quy:	11
e.	Chạy chương trình excel_to_xml.py:	14
2.	Cách thức chuyển từ XML về lại XLSX:	15
IV.	<i>References:</i>	16

I. Dẫn nhập về định dạng XML:

1. XML là gì? Tác dụng của file XML:

XML (Extensible Markup Language) là ngôn ngữ đánh dấu được tạo ra bởi World Wide Web Consortium (W3C) để xác định cú pháp mã hoá tài liệu để con người và máy có thể đọc được. Đây là một dạng ngôn ngữ đánh dấu, có chức năng truyền dữ liệu và mô tả nhiều loại dữ liệu khác nhau. Nó thực hiện điều này thông qua việc sử dụng thẻ xác định cấu trúc tài liệu cũng như cách tài liệu được lưu trữ và vận chuyển.

Mục đích chính của XML là đơn giản hoá việc chia sẻ dữ liệu giữa các platform và các hệ thống được kết nối với mạng Internet. Chính vì vậy, XML có tác dụng rất lớn trong việc chia sẻ, trao đổi dữ liệu giữa các hệ thống.

Có thể dễ dàng so sánh XML với một ngôn ngữ đánh dấu khác là Hypertext Markup Language (Ngôn ngữ đánh dấu siêu văn bản - HTML) được sử dụng để mã hoá các trang web. HTML sử dụng một tập hợp các ký hiệu đánh dấu được xác định trước mô tả định dạng nội dung trên một trang web.

Tuy nhiên, điểm khác nhau là XML có thể mở rộng được, nó không có ngôn ngữ đánh dấu được xác định trước như HTML. Thay vào đó, XML cho phép người dùng tạo biểu tượng đánh dấu riêng để mô tả nội dung, tạo một biểu tượng không giới hạn và tự định nghĩa. Đặc biệt, HTML là ngôn ngữ tập trung vào việc trình bày nội dung, trong khi XML là ngôn ngữ mô tả dữ liệu được sử dụng để lưu trữ dữ liệu.

XML thường được sử dụng làm cơ sở cho các định dạng tài liệu khác như: RSS, ATOM, Microsoft .NET...

Đây là một ví dụ về cú pháp của XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Hình 1.1.1. Ví dụ về cú pháp XML

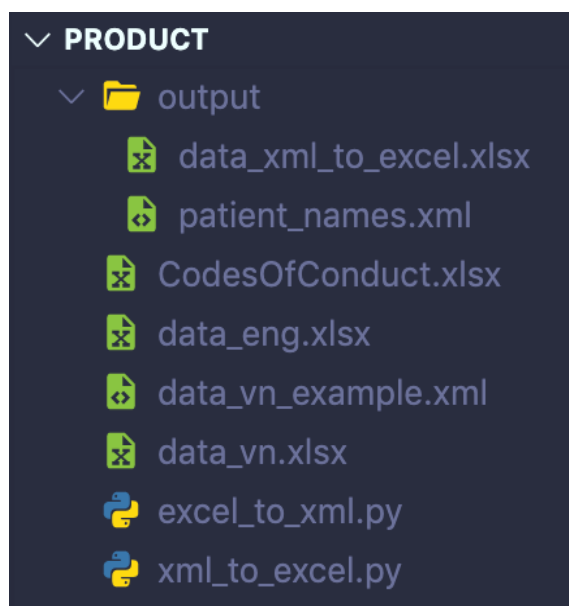
2. Đặc điểm của XML:

- XML được dùng cho dữ liệu có cấu trúc.
- Về trực quan, XML khá giống với HTML.
- Tuy là văn bản, nhưng XML không phải dùng để đọc.
- Cấu trúc file XML thường rất dài.

- XML được coi là cầu nối đưa HTML tới XHTML và là nền tảng cho RDF và Web mã hoá.
- XML là một module.
- XML miễn phí bản quyền, platform độc lập và được hỗ trợ rất tốt.

II. Cấu tạo thư mục & thư viện cần có:

1. Cấu tạo thư mục:



Truy cập repo của project tại [link này](#).

Dựa trên [trang web](#) chứa quy định chuẩn và định dạng dữ liệu trong quản lý y tế của *thuvienphapluat.vn*, tác giả đã soạn lại một số tiêu chuẩn trong file excel *CodesOfConduct.xlsx* ở thư mục ngoài *product*. File này chứa những tiêu chuẩn của các trường của dữ liệu, chia làm 4 sheets, minh hoạ như hình 1.2.

Ở những file trong thư mục con *product*, chứa file *data_vn.xlsx* là file excel để trích xuất dữ liệu. File *data_eng.xlsx* là file test dựa trên quy chuẩn của [FHIR Specification](#).

Hình 2.1.1. Cấu trúc thư mục

File *excel_to_xml.py* là file code python để đọc file .xlsx và xuất dữ liệu ra định dạng .xml, data đó cuối cùng được lưu trong file *patient_names.xml* trong foler con *output*.

File *xml_to_excel.py* làm tác vụ ngược lại với file trên là đọc file *patient_names.xml* và xuất ra lại file *data_xml_to_excel.xlsx*.

Ngoài ra, lúc tìm hiểu tác giả có soạn thử một file .xml mẫu là file *data_vn_example.xml* ở thư mục ngoài, kết quả file .xml ra được định hướng theo file này.

Bảng 1. Chỉ tiêu tổng hợp khám bệnh, chữa bệnh BHYT

STT	Chỉ tiêu	Kiểu dữ liệu	Kích thước tối đa	Diễn giải
1	MA_LK	Chuỗi	100	Mã đợt điều trị duy nhất (dùng để liên kết giữa bảng tổng hợp (bảng 1) và các bảng chi tiết (từ
2	STT	Số	10	STT tăng từ 1 đến hết trong 1 lần gửi dữ liệu.
3	MA_BN	Chuỗi	100	Mã số bệnh nhân quy định tại cơ sở khám bệnh, chữa bệnh.
4	HO_TEN	Chuỗi	255	Họ và tên người bệnh
5	NGAY_SINH	Chuỗi	8	Ngày sinh ghi trên thẻ gồm 8 ký tự; 4 ký tự năm + 2 ký tự tháng + 2 ký tự ngày (nếu không có)
6	GIOI_TINH	Số	1	Giới tính; Mã hóa (1: Nam; 2: Nữ; 3: Chưa xác định)
7	DIA_CHI	Chuỗi	1024	Ghi địa chỉ theo địa chỉ trên thẻ BHYT hoặc nơi cư trú hiện tại của người bệnh: số nhà (nếu có
8	MA_THE	Chuỗi	n	- Mã thẻ BHYT do cơ quan BHXH cấp - Trường hợp chưa có thẻ BHYT nhưng vẫn được hưởng quyền lợi BHYT, Ví dụ: trẻ em, người ghép tạng,... Ví dụ: TE101KT00000011 (Mã thẻ tạm cho trẻ em thứ 11 đến khám, giấy khai sinh/chứng sinh cấp tại Hà N - Trường hợp trong thời gian điều trị, người bệnh được cấp thẻ BHYT mới có thay đổi thông tin liên quan đ
9	MA_DKBD	Chuỗi	n	Mã cơ sở khám bệnh, chữa bệnh nơi người bệnh đăng ký ban đầu ghi trên thẻ BHYT, gồm có - Trường hợp trong thời gian điều trị, người bệnh được cấp thẻ BHYT mới có thay đổi thông tin - Trường hợp chưa có thẻ BHYT: Ghi mã đơn vị hành chính của tỉnh/TP + 000. Ví dụ: Hà Nội t
10	GT_THE_TU	Chuỗi	n	Thời điểm thẻ có giá trị gồm 8 ký tự; 4 ký tự năm + 2 ký tự tháng + 2 ký tự ngày - Trường hợp trong thời gian điều trị, người bệnh được cấp thẻ BHYT mới có thay đổi thông tin liên quan đ - Trường hợp chưa có thẻ BHYT: Thay thời điểm thẻ có giá trị bằng ngày người bệnh đến khám bệnh, chữa
11	GT.THE_DEN	Chuỗi	n	Thời điểm thẻ hết giá trị gồm 8 ký tự; 4 ký tự năm + 2 ký tự tháng + 2 ký tự ngày - Trường hợp trong thời gian điều trị, người bệnh được cấp thẻ BHYT mới có thay đổi thông tin - Trường hợp chưa có thẻ BHYT: Thay thời điểm thẻ hết giá trị bằng ngày người bệnh ra viện
12	MIEN_CUNG_CT	Chuỗi	8	- Thời điểm người bệnh bắt đầu được hưởng miễn cùng chi trả theo giấy xác nhận của cơ quan BHXH, gồm Ví dụ: ngày 31/03/2017 được hiển thị là: 20170331 - Nếu không có giấy xác nhận miễn cùng chi trả của cơ quan BHXH thì để trống
13	TEN_BENH	Chuỗi	n	Ghi đầy đủ các chẩn đoán được ghi trong hồ sơ, bệnh án

Hình 2.1.2. Nội dung minh họa của file CodeOfConduct.xlsx

```

1  <Total_patients>
2    <patient>
3      <name>
4        <use value='official'></use>
5        <family value='Thai'></family>
6        <given value='Quang'></given>
7        <given value='Nguyen'></given>
8      </name>
9      <name>
10       <use value='nickname'></use>
11       <given value='Jack'></given>
12     </name>
13     <telecom>
14       <system value='phone'></system>
15       <value value='+84913981394'></value>
16       <use value='mobile'></use>
17       <rank value='1'></rank>
18     </telecom>
19     <birthdate>
20       <value value='2000-12-03'></value>
21     </birthdate>
22     <address>
23       <use value='home'></use>
24       <text value='33 Ngo Thoi Nham, Thuan Hoa, Hue, Thua Thien Hue 49000'></text>
25       <line value='33 Ngo Thoi Nham'></line>
26       <ward value='Thuan Hoa'></ward>
27       <city value='Hue'></city>
28       <province value='Thua Thien Hue'></province>
29       <postalcode value='49000'></postalcode>
30       <period>
31         <start value='2000'></start>
32         <end value='2018'></end>
33       </period>
34     </address>

```

Hình 2.1.3. Nội dung của file data_vn_example.xml

2. Các thư viện cần có:

- [openpyxl](#): thư viện dùng để đọc/ghi file Excel theo những định dạng xlsx/xlsm/xltx/xltn.
- [yattag](#): thư viện dùng để tạo ra file HTML hoặc XML bằng code Python.
- [datetime](#): thư viện để đọc và tính toán ngày tháng trong Python.
- [BeautifulSoup](#): dùng để đọc file xml.
- [queue](#): dùng để sử dụng *PriorityQueue* nhằm tạo hàng ưu tiên sẽ được đề cập ở phần sau.

III. Diễn giải code:

1. Chuyển từ EXCEL về XML:

a. Cách thức đọc file EXCEL:

Trước tiên ta import hàm *load_workbook()* từ thư viện *openpyxl* và gọi hàm với đối số là tên của file excel chứa dữ liệu:

```
from openpyxl import load_workbook
wb = load_workbook("data_vn.xlsx")
ws = wb.worksheets[0]
```

Tiếp đến ta import thư viện *yattag* để xuất định dạng xml:

```
from yattag import Doc, indent
# Create Yattag doc, tag and text objects
doc, tag, text = Doc().tagtext()
```

Class *yattag.Doc* hoạt động như cách ta liên kết các chuỗi lại với nhau, ví dụ đơn giản như hình dưới:

```
mylist = []
mylist.append('Everybody')
mylist.append('likes')
mylist.append('pandas.')
mystring = ' '.join(mylist) # mystring contains "Everybody likes pandas."
```

Hình 3.1.1. Cách thức hoạt động của class *yattag.Doc*

	A	B	C
1	NODEOPEN_PATIENT		
2	NODEOPEN_NAME		
3	NAME_USE	official	official
4	NAME_FAMILY	Thai	Thai
5	NAME_GIVEN	Quang	Minh
6	NAME_GIVEN	Nguyen	Khue
7	NODECLOSE_NAME		
8	NODEOPEN_NAME		
9	NAME_USE	nickname	nickname
10	NAME_GIVEN	Jack	Horse
11	NODECLOSE_NAME		
12	NODEOPEN_TELECOM		
13	TELECOM_SYSTEM	phone	phone
14	TELECOM_VALUE	+84913981394	+84914002126
15	TELECOM_USE	mobile	mobile
16	TELECOM_RANK	1	1
17	NODECLOSE_TELECOM		
18	NODEOPEN_BIRTHDATE		
19	BIRTHDATE_VALUE	2000-12-03	2000-12-04
20	NODECLOSE_BIRTHDATE		
21	NODEOPEN_ADDRESS		
22	ADDRESS_USE	home	home
23	ADDRESS_TEXT	a, Hue, Thua Thien Hue 49000	n Hoa, Hue, Thua Thien Hue 49000
24	ADDRESS_LINE	33 Ngo Thoi Nham	34 Ngo Thoi Nham
25	ADDRESS_WARD	Thuan Hoa	Thuan Hoa
26	ADDRESS_CITY	Hue	Hue
27	ADDRESS_PROVINCE	Thua Thien Hue	Thua Thien Hue
28	ADDRESS_POSTALCODE	49000	49000
29	NODEOPEN_PERIOD		
30	PERIOD_START	2000	2005
31	PERIOD_END	2018	2018
32	NODECLOSE_PERIOD		
33	NODECLOSE_ADDRESS		
34	NODEOPEN_ADDRESS		
35	ADDRESS_USE	temporary	temporary
36	ADDRESS_TEXT	ang, Di An, Binh Duong 75000	h Thang, Di An, Binh Duong 75001
37	ADDRESS_LINE	Samsora Riverside	Samsora Riverside
38	ADDRESS_WARD	Binh Thang	Binh Thang
39	ADDRESS_CITY	Di An	Di An
40	ADDRESS_PROVINCE	Binh Duong	Binh Duong

Hình 3.1.2. Nội dung file data_vn.xlsx

Tạo mẫu file *data_vn.xlsx* chứa nội dung là các trường (cột A) và các thông số tương ứng của từng bệnh nhân (từ cột B trở đi).

Trong đó, ở những hàng được tô màu xanh (*NODEOPEN*) và những hàng được tô màu cam (*NODECLOSE*) nhằm phục vụ việc mở và đóng các *tag* trong file .xml. Những ký hiệu đóng mở này cũng tương tự như những brackets ‘{’ hay ‘}’ trong ngôn ngữ C++,

đánh dấu việc bắt đầu và kết thúc của một hàm. Về việc đọc những bracket này tác giả sẽ đề cập ở phần sau.

Như ở hình dưới, sẽ có các tag mà trong chúng chứa những tag con, tag cha là *name* và tag con là *use*, *family*, *given*, *given*. Các tag con không chứa thêm tag con trong chúng và được gọi là *self-closing tag*. Qua đó, nếu muốn nhét các tag con vào tag cha, ta dùng 2 ký hiệu đóng mở *NODEOPEN* và *NODECLOSE*.

```
<name>
  <use value="official" />
  <family value="Thai" />
  <given value="Quang" />
  <given value="Nguyen" />
</name>
```

Hình 3.1.3. Chú thích về sự sắp xếp các tags

b. Tạo class XMLNode:

Tác giả sử dụng 1 class gọi là *XMLNode* để gán kiểu cho các tag cha như tag *name* ở bên trên. Về ý tưởng, khi input là một file excel như file *data_vn.xlsx*, code sẽ quét qua một lượt để **đánh dấu những brackets đóng và mở các tag cha**, đồng thời lưu số dòng tương ứng của bracket lại thành điểm đầu, điểm cuối và tên (các tính chất) của object thuộc class *XMLNode*, object ở đây ví dụ là trường *name* như hình 3.1.3. Ta khai báo class *XMLNode* như sau:

```
class XMLNode(object):
    def __init__(self, start=None, end=None, text=None):
        self.start = start
        self.end = end
        self.text = text
```

Sau khi đã đánh dấu hết được các tag có khả năng làm cha này, ta sẽ dùng **hàm đệ quy** để quét qua dữ liệu (được lưu từ cột B, C, ... trở đi của file *data_vn.xlsx*). Ví dụ khi quét đến một hàng trong file excel mà đó là điểm bắt đầu của một tag cha, ta mở tag đó và gọi lại hàm quét để quét tiếp các tag con. Sau khi đã quét hết các tag con, con trỏ PC sẽ **tự động nhảy về lại nơi hàm con được gọi** trong hàm cha. Đây là ý tưởng sơ khởi việc quét file EXCEL sử dụng đệ quy, tác giả sẽ cố gắng trình bày rõ hơn ở các phần sau. Trước mắt, **ta tạo được một class gọi là XMLNode, các object trong class này có 3 tính chất: số thứ tự dòng mà tag đó mở ra, số thứ tự dòng mà tag đóng lại, và tên của tag.**

c. Scan file EXCEL:

Ta định nghĩa hàm `scan_excelFile()` như sau:

```
def scan_excelFile():
    for col in ws.iter_cols(min_col=1, max_col=1, min_row=BEGIN_ROW, max_row=END_ROW):
        col = [cell.value for cell in col]
        for _line, _data in enumerate(col):
            _code = _data.split('_')
            if _code[0] == "NODEOPEN":
                _node = XMLNode(start=_line, text=_code[1].lower())
                XMLNode_stack.append(_node)
            elif _code[0] == "NODECLOSE":
                _node = XMLNode_stack.pop()
                _node.end = _line
                XML_list.put((_node.start, _node.end, _node.text, _node))
            else:
                XML_list.put((_line, -1, _code[1].lower(), _data))
    return _node
```

Trong hàm này, ta thực hiện việc quét qua từng hàng của cột A trong file excel (cột chứa tên của các trường và các ký hiệu *NODEOPEN*, *NODECLOSE*). Do chỉ có một cột A nên đổi số *min_col* và *max_col* của hàm `ws.iter_cols()` đều bằng 1. Và *min_row* và *max_row* sẽ được gán bằng số thứ tự hàng bắt đầu và kết thúc của danh sách các trường dữ liệu. Sau đó mảng 1 chiều *col* sẽ được lưu dữ liệu của cả cột A thông qua truy xuất giá trị từng ô trong excel bằng cú pháp *cell.value*.

Như vậy ta có từng phần tử của mảng *col* chính là giá trị từng hàng của cột A, ta sẽ thêm *index* (số đếm) cho các phần tử này bằng hàm `enumerate()`, nôm na là thêm một cột số thứ tự vào cột dữ liệu. Cách thức hoạt động của hàm `enumerate()` được minh họa như hình dưới:

```
# Python program to illustrate
# enumerate function
l1 = ["eat", "sleep", "repeat"]
s1 = "geek"
```

```
# creating enumerate objects
obj1 = enumerate(l1)
obj2 = enumerate(s1)
```

```
print "Return type:", type(obj1)
print list(enumerate(l1))
```

```
Return type: < type 'enumerate' >
[(0, 'eat'), (1, 'sleep'), (2, 'repeat')]
```

Hình 3.1.4. Ví dụ về hàm `enumerate()`

Tiếp theo ta sẽ quét qua từng hàng của mảng `enumerate(col)` bằng vòng lặp *for*, với biến *_line* là *số thứ tự dòng* (bắt đầu từ 0), và *_data* chính là *dữ liệu các hàng trong cột A của file excel*.

2	NODEOPEN_NAME
3	NAME_USE
4	NAME_FAMILY
5	NAME_GIVEN
6	NAME_GIVEN
7	NODECLOSE_NAME

Quay lại với file excel, tác giả đã cố tình lồng tên của tag cha vào điểm mở tag, được ngăn cách bằng ký tự '_'. *NODEOPEN* là lệnh để mở tag, *NAME* chính là tên của tag cần được mở. Chính vì vậy trong hàm *scan_excelFile()* bên trên, ta **chia chuỗi này thành 2 chuỗi riêng bằng hàm split()** và lưu vào vector *_code*.

Nếu *_code[0]* bằng “*NODEOPEN*”, tức là ta đang quét đến điểm mở tag, thì ta tạo một XMLNode object với **điểm bắt đầu (start) bằng số thứ tự dòng đang quét (_line), tên của tag là _code[1]** được in thường bằng hàm *lower()*.

Do ta mới quét tới điểm mở tag, **chưa tới điểm đóng** nên chưa thể điền đầy đủ thông tin cho tag cha này. Do vậy ta **sử dụng stack XMLNode_stack theo quy luật Last In First Out để lưu tạm những object này, chờ đến khi gặp điểm kết thúc thì sẽ cập nhật vào tính chất của chúng**.

Sau khi quét tới điểm mở, ta đẩy object thuộc class XMLNode đã được tạo vào stack (lệnh *append()*). Tới thời điểm đóng, tức là *_code[0]* bằng “*NODECLOSE*”, ta sẽ pop object ở trên cùng ra (lệnh *pop()*) để **gán thêm số thứ tự dòng kết thúc cho nó (end)**. Sử dụng stack có thể giải quyết được bài toán khi ta mở tag trong tag, sao cho giá trị trên cùng của stack sẽ luôn là object XMLNode nhỏ nhất, để khi đóng ta sẽ đóng cái ta vừa mở. Đồng thời, để đóng hết thì ta sẽ phải pop hết tất cả các giá trị trong stack, tránh trường hợp có lỗi xảy ra.

Sau khi object XMLNode đã có đủ 3 tính chất start, end, text, ta đặt nó vào trong một vector 2 chiều *XML_list* bằng lệnh *put()*. Đây không phải là một vector 2 chiều bình thường mà chính là một **hàng ưu tiên (PriorityQueue)** đã được tác giả khởi tạo trước đó. Cách hoạt động của PriorityQueue được minh hoạ như sau:

```
from queue import PriorityQueue

q = PriorityQueue()

q.put((2, 'code'))
q.put((1, 'eat'))
q.put((3, 'sleep'))

while not q.empty():
    next_item = q.get()
    print(next_item)
```

Result:

```
# (1, 'eat')
# (2, 'code')
# (3, 'sleep')
```

Hình 3.1.5. Minh hoạ class PriorityQueue

PriorityQueue sẽ tự động sắp xếp các mảng ta đưa vào để **giá trị đầu mỗi mảng xếp theo thứ tự**. Ở đây ý đồ của tác giả là sắp xếp các mảng thông tin các trường (điểm bắt đầu, điểm kết thúc, tên, loại) theo điểm bắt đầu. Vì các điểm bắt đầu các trường đều là khác nhau và là số đếm, ta *không cần quan tâm đến lỗi Type Conflict* có thể diễn ra khi *PriorityQueue* tự động so sánh các giá trị kế tiếp khi giá trị đầu bằng nhau.

Ngoài ra, nếu ta quét đến một dòng mà không phải điểm đóng hay mở của một tag cha thì đây chính là tag con self-closing theo như quy chuẩn. Các tag con này chính là những tag chứa dữ liệu trong 'value', có thể thấy rằng trong file *data_vn.xlsx* từ cột B trở đi, vị trí các dòng tag con này sẽ chứa dữ liệu, còn vị trí của brackets sẽ không có dữ liệu nào. Khi quét đến các dòng tag con này, ta cũng tương tự thêm vào *XML_list* (một *PriorityQueue*) giá trị bắt đầu, tên, giá trị kết thúc cho bằng -1.

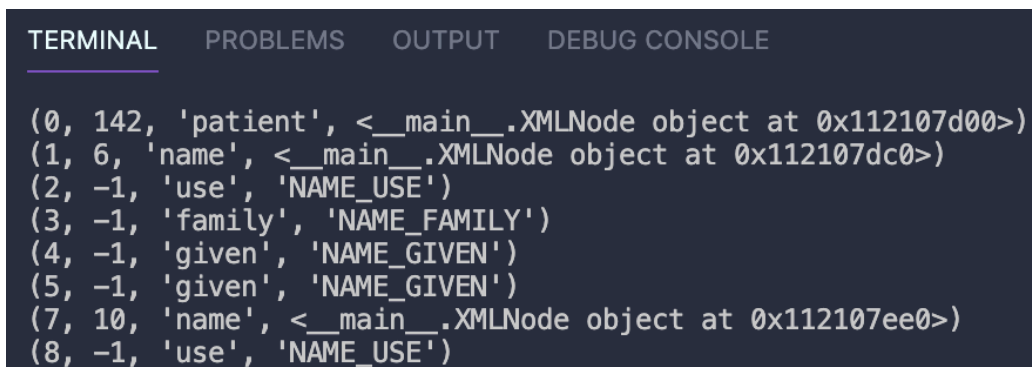
```
XML_list.put((_node.start, _node.end, _node.text, _node))
else:
    XML_list.put((_line, -1, _code[1].lower(), _data))
```

Lưu ý rằng mỗi hàng của *XML_list* sẽ có 4 thông số: start, end, text, và object *XMLNode* nếu là tag cha hoặc chuỗi tên hoàn chỉnh của tag trong trường hợp là tag con. Thông số cuối cùng này để giúp ta phân biệt liệu một tag có phải là object của *XMLNode* hay không thông qua lệnh *isinstance()* sẽ được đề cập ở phần sau.

Như vậy sau khi gọi hàm này, ta được một hàng ưu tiên lưu các object *XMLNode* tương trưng cho các brackets và các trường dữ liệu có giá trị (value) từ file excel. Ta lưu lại hàng ưu tiên vào một vector 2 chiều khác tên *XML_list2* cho dễ thao tác như sau:

```
while not XML_list.empty():
    XML_list2.append(XML_list.get())
```

Chạy hàm này *scan_excelFile()* được kết quả tương tự như hình dưới:



```

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE
(0, 142, 'patient', <__main__.XMLNode object at 0x112107d00>)
(1, 6, 'name', <__main__.XMLNode object at 0x112107dc0>)
(2, -1, 'use', 'NAME_USE')
(3, -1, 'family', 'NAME_FAMILY')
(4, -1, 'given', 'NAME_GIVEN')
(5, -1, 'given', 'NAME_GIVEN')
(7, 10, 'name', <__main__.XMLNode object at 0x112107ee0>)
(8, -1, 'use', 'NAME_USE')
```

Hình 3.1.6. Kết quả hàm *scan_excelFile()*

d. Đọc và lưu xml các trường dữ liệu bằng đệ quy:

Như ở phần tạo class *XMLNode* đã đề cập, ta sẽ dùng phương pháp đệ quy để quét qua dữ liệu của file excel. Định nghĩa thêm một hàm *printNodes()* trong class *XMLNode* như sau:

```

def printNodes(self, data):
    global lineCounter
    global offset
    while ((lineCounter+offset) == XML_list2[lineCounter][0]) and
(lineCounter < len(XML_list2)):
        if isinstance(XML_list2[lineCounter][3], XMLNode):
            with tag(XML_list2[lineCounter][2]):
                lineCounter += 1
                XML_list2[lineCounter-1][3].printNodes(data)
        elif isinstance(XML_list2[lineCounter][3], str):
            if data[lineCounter+offset] is None:
                tmp_value = ''
            else:
                tmp_value = data[lineCounter+offset]
            doc.stag(XML_list2[lineCounter][2], '', value=tmp_value)
            lineCounter += 1
    offset += 1

```

Hai biến *lineCounter* và *offset* là hai biến được khai báo toàn cục bằng 0 và được sử dụng trong hàm, mỗi khi quét từng cột B, C... hai biến này sẽ được đặt trở về giá trị 0. Biến *lineCounter* dùng để đếm từng hàng trong *XML_list2* và tương ứng với các hàng bên file *data_vn.xlsx*. Tuy nhiên, vì với các tag cha có 2 brackets nhưng ta chỉ lưu một hàng trong vector *XML_list2*, trong khi bên excel thì vẫn có 2 hàng, ta dùng một biến nữa là *offset* để cộng lên mỗi lần đến điểm đóng của tag cha đó, như vậy ta có thể *tương ứng các dòng của cả XML_list2 và file data_vn.xlsx thông qua lineCounter và lineCounter+offset*.

Ta sẽ đi lần lượt từ lúc bắt đầu đọc một cột. Lúc đầu, *lineCounter* = 0 và *offset* = 0. *Giá trị lineCounter+offset do đó sẽ bằng với giá trị điểm bắt đầu của bracket 'patient'* là 0. Cú pháp *with tag()* của thư viện *yattag* cho ta mở một tag cha, ta có thể đặt các tag con vào trong, khi kết thúc hàm *with*, *yattag* sẽ thực hiện việc đóng tag cha này lại.

```

(0, 142, 'patient', <__main__.XMLNode object at 0x112107d00>)
(1, 6, 'name', <__main__.XMLNode object at 0x112107dc0>)
(2, -1, 'use', 'NAME_USE')
(3, -1, 'family', 'NAME_FAMILY')

```

Hình 3.1.7. Minh họa việc đọc file đệ quy

Khi muốn mở tag trong tag, trong trường hợp này chính là mở tag 'name' sau khi vừa mở tag 'patient', ta sẽ gọi lại hàm *printNodes()* lần nữa. Do đó hàm *with tag()* của 'patient' chưa kết thúc, khi đã làm việc xong với tag 'name', ta sẽ đóng nó (thoát khỏi hàm *with tag()*) rồi con trở PC chương trình mới trở lại vị trí đã gọi hàm *printNodes()* mở tag 'name', hàm tiếp tục quét các tag cùng thế hệ với 'name' tới khi hết mới đóng tag 'patient'.

Ta check xem một hàng trong cột B có phải *XMLNode* không thông qua hàm *isinstance()* với đối số là thông số cuối cùng của *XML_list2*. Nếu là một object thuộc *XMLNode*, như trên đã đề cập, ta sẽ **gọi lại hàm *printNodes()***, đồng thời cộng *lineCounter* lên 1. Hàm *printNodes()* vừa được gọi lại check *lineCounter+offset* có bằng *XML_list2[lineCounter][0]* không. Kết quả chúng bằng nhau và bằng 1. Ở hàng thứ 1 của *XML_list2* chứa một object *XMLNode* là 'name', do đó ta lại cộng *lineCounter* lên 1 và gọi lại hàm *printNodes()* lần nữa.

Hàm *printNodes()* vừa được gọi sau đó lại vượt qua hàm *while* để check xem hàng đó thuộc kiểu gì. Kết quả là *XML_list2[lineCounter][3]* không phải là một *XMLNode* mà chỉ là một chuỗi bình thường, tương ứng với hàng chứa dữ liệu trong file *data_vn.xlsx*. Lưu ý rằng như đã đề cập, ta truy xuất data (là dữ liệu trong file excel) thông qua index là *lineCounter+offset*. Để tránh trường hợp không đọc được kiểu *None*, ta thay nó bằng ký tự trống. Sau đó ta thêm vào file .xml một self-closing tag có value chính là dữ liệu của hàng từ file excel, và tên lấy từ *XML_list2[lineCounter][2]*. *lineCounter* được cộng lên 1. Theo hàm *while*, ta xuất các tag con của tag 'name' một cách tương tự.

```
(0, 142, 'patient', <__main__.XMLNode object at 0x112107d00>)
(1, 6, 'name', <__main__.XMLNode object at 0x112107dc0>)
(2, -1, 'use', 'NAME_USE')
(3, -1, 'family', 'NAME_FAMILY')
(4, -1, 'given', 'NAME_GIVEN')
(5, -1, 'given', 'NAME_GIVEN')
(7, 10, 'name', <__main__.XMLNode object at 0x112107ee0>)
(8, -1, 'use', 'NAME_USE')
```

Khi chuyển từ hàng 5 sang hàng 7, *lineCounter+offset* không bằng *XML_list2[lineCounter][0]* nên sẽ thoát khỏi hàm *while*, đồng thời *offset* được cộng lên 1. Đây chính là điểm kết thúc của một tag cha (cụ thể là tag name). Sau đó *lineCounter+offset* lại bằng *XML_list2[lineCounter][0]* và bằng 7. Hàm tiếp tục mở tag 'name' thứ hai và tiếp tục vai trò như hồi nãy. Cứ như vậy cho đến khi hết cột, ta sẽ xuất ra được file .xml minh hoạ như sau:

```
<patient>
  <name>
    <use value="official" />
    <family value="Thai" />
    <given value="Quang" />
    <given value="Nguyen" />
  </name>
  <name>
    <use value="nickname" />
    <given value="Jack" />
  </name>
```

Hình 3.1.7. Minh hoạ kết quả hàm *printNodes()*

e. Chạy chương trình excel to xml.py:

Ta định nghĩa thêm hàm *export_xmlFile()* để quét từng cột của file excel và gọi từng hàm *printNodes()*.

```
def export_xmlFile(rootNode):
    global lineCounter
    global offset
    with tag ("totalpatients"):
        for col in ws.iter_cols(min_col=BEGIN_COL, max_col=END_COL,
min_row=BEGIN_ROW, max_row=END_ROW):
            col = [cell.value for cell in col]
            lineCounter = 0
            offset = 0
            rootNode.printNodes(col)
```

Sau đó xuất ra file *patient_names.xml* bằng lệnh “w”:

```
patientNode = scan_excelFile()
export_xmlFile(patientNode)
result = indent(
    doc.getvalue(),
    indentation = '    '
)
with open("output/patient_names.xml", "w") as f:
    f.write(result)
```

Ta được file *patient_names.xml* trong thư mục *output* giống với file mong muốn:

```
output > patient_names.xml > totalpatients > patient > birthdate > value
1  <?xml version="1.0" encoding="UTF-8"?>
2  <totalpatients>
3    <patient>
4      <name>
5        <use value="official" />
6        <family value="Thai" />
7        <given value="Quang" />
8        <given value="Nguyen" />
9      </name>
10     <name>
11       <use value="nickname" />
12       <given value="Jack" />
13     </name>
14     <telecom>
15       <system value="phone" />
16       <value value="+84913981394" />
17       <use value="mobile" />
18       <rank value="1" />
19     </telecom>
20     <birthdate>
21       <value value="2000-12-03" />
22     </birthdate>
23     <address>
24       <use value="home" />
25       <text value="33 Ngo Thoi Nham, Thuan Hoa, Hue, Thua Thien Hue 49000" />
26       <line value="33 Ngo Thoi Nham" />
27       <ward value="Thuan Hoa" />
28       <city value="Hue" />
29       <province value="Thua Thien Hue" />
30       <postalcode value="49000" />
31       <period />
32     </address>
33   </patient>
34 </totalpatients>
```

Hình 3.1.8. Dữ liệu lưu trong file *patient_names.xml*

2. Cách thức chuyển từ XML về lại XLSX:

Với file *output/patient_names.xml* có được từ việc chạy file *excel_to_xml.py*, ta có thể tiếp đến đọc ngược lại file .xml về lại .xlsx bằng thư viện BeautifulSoup.

Ta định nghĩa một hàm *readFile()* như sau để đọc dữ liệu vào nhờ *BeautifulSoup*:

```
def readFile(filename):
    if not os.path.exists(filename):
        print("Cannot find .xml file!")
        os._exit(0)
        return
    with open(filename, 'r') as f:
        data = f.read()
    Bs_data = BeautifulSoup(data, "xml")
```

Ta tạo một vector 2 chiều *mdlist* để lưu các dữ liệu, trong đó hàng 0 của vector lưu tên của các trường.

```
for _node in Bs_data.find("patient").findChildren(recursive=True):
    if _node.get('value') is not None:
        sub_mdlist.append(_node.parent.name.upper()+'_'+_node.name.upper())
mdlist.append(sub_mdlist)
```

Với tính chất của dữ liệu, chỉ những tag con mới có value cho nên ta có thể quét qua các tag con của một tag '*patient*', lấy tất cả các tag có value và thêm tên của tag đó vào *mdlist*.

Ở các hàng sau, ta sẽ thêm các value vào *mdlist* thông qua tất cả các tag '*patient*' bằng lệnh *find_all()*:

```
for _patient in Bs_data.find_all("patient"):
    sub_mdlist = []
    for _node in _patient.findChildren(recursive=True):
        if _node.get('value') is not None:
            sub_mdlist.append(_node.get('value'))
    mdlist.append(sub_mdlist)
```

Khi đã có một vector hai chiều là *mdlist[]*, tiếp đến ta bỏ nó vào file excel mới trong thư mục *output* bằng hàm *to_Excel()*:

```
def to_Excel(mdlist):
    wb = Workbook()
```



```

ws = wb.active
for i,row in enumerate(mdlist):
    for j,value in enumerate(row):
        ws.cell(row=i+1, column=j+1).value = value
newfilename = os.path.abspath("./output/data_xml_to_excel.xlsx")
wb.save(newfilename)
print("Process completed")
return

```

Dùng hàm *enumerate()* để đánh số từng dòng của *mdlist[]* và quét các số đó thông qua biến *i*, quét dữ liệu thông qua biến *row*. Dùng hàm *enumerate()* tương tự với mảng *row* (tức là từng hàng của *mdlist[]*), lưu các số để đánh thứ tự vào biến *j*. Sau là một ví dụ của hàm *enumerate()*:

```

# Python program to illustrate
# enumerate function
l1 = ["eat", "sleep", "repeat"]
s1 = "geek"

```

```

Return type: < type 'enumerate' >
[(0, 'eat'), (1, 'sleep'), (2, 'repeat')]

```

```

# creating enumerate objects
obj1 = enumerate(l1)
obj2 = enumerate(s1)

```

Hình 3.2.1. Ví dụ về hàm *enumerate()*

```

print "Return type:",type(obj1)
print list(enumerate(l1))

```

Qua đó, ta có biến *i* chỉ số hàng và biến *j* chỉ số cột trong file excel, trong file excel thì số đếm bắt đầu từ 1 nên ta cộng *i* và *j* với 1. Gán giá trị của từng cell bằng hàm *cell().value*. Cuối cùng ta save lại file excel mới trong thư mục *output*:

```

result = readFile("output/patient_names.xml")
if result:
    to_Excel(result)

```

	A	B	C	D	E	F	G
1	NAME_USE	NAME_FAM	NAME_GIVI	NAME_GIVI	NAME_USE	NAME_GIVI	TELECOM_S
2	official	Thai	Quang	Nguyen	nickname	Jack	phone
3	official	Thai	Minh	Khue	nickname	Horse	phone

Hình 3.2.2. Kết quả file *data_xml_to_excel.xlsx*

IV. References:

<https://stackoverflow.com/>

<https://www.javatpoint.com/xml-tutorial>

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>