**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. Dẫn nhập về định dạng XML: 3](#_Toc58578851)

[1. XML là gì? Tác dụng của file XML: 3](#_Toc58578852)

[2. Đặc điểm của XML: 3](#_Toc58578853)

[II. Cấu tạo thư mục & thư viện cần có: 4](#_Toc58578854)

[1. Cấu tạo thư mục: 4](#_Toc58578855)

[2. Các thư viện cần có: 6](#_Toc58578856)

[III. Diễn giải code: 6](#_Toc58578857)

[1. Chuyển từ EXCEL về XML: 6](#_Toc58578858)

[**a.** **Cách thức đọc file EXCEL:** 6](#_Toc58578859)

[**b.** **Tạo class XMLNode:** 8](#_Toc58578860)

[**c.** **Scan file EXCEL:** 9](#_Toc58578861)

[**d.** **Đọc và lưu xml các trường dữ liệu bằng đệ quy:** 11](#_Toc58578862)

[**e.** **Chạy chương trình excel\_to\_xml.py:** 14](#_Toc58578863)

[2. Cách thức chuyển từ XML về lại XLSX: 15](#_Toc58578864)

[IV. References: 16](#_Toc58578865)

1. **Dẫn nhập về định dạng XML:**
2. **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.

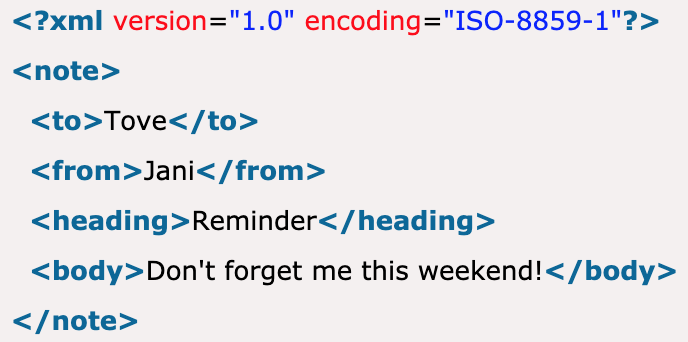
Mục đích chính của XML là đơn giản hoá việc chia sẻ dữ liệu giữa các platform và các hệ thống được kết nối với mạng Internet. Chính vì vậy, XML có tác dụng rất lớn trong việc chia sẻ, trao đổi dữ liệu giữa các hệ thống.

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:

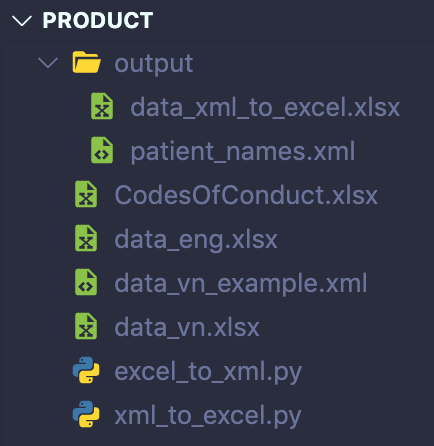


*Hình 1.1.1. Ví dụ về cú pháp XML*

1. **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.

1. **Cấu tạo thư mục & thư viện cần có:**
2. **Cấu tạo thư mục:**

Truy cập repo của project tại [link này](https://github.com/TQN-BACHKHOA/201_XLSTH_MR).

Dựa trên [trang web](https://thuvienphapluat.vn/van-ban/bao-hiem/Quyet-dinh-4210-QD-BYT-2017-du-lieu-dau-ra-trong-thanh-toan-chi-phi-kham-chua-benh-y-te-361955.aspx) 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.

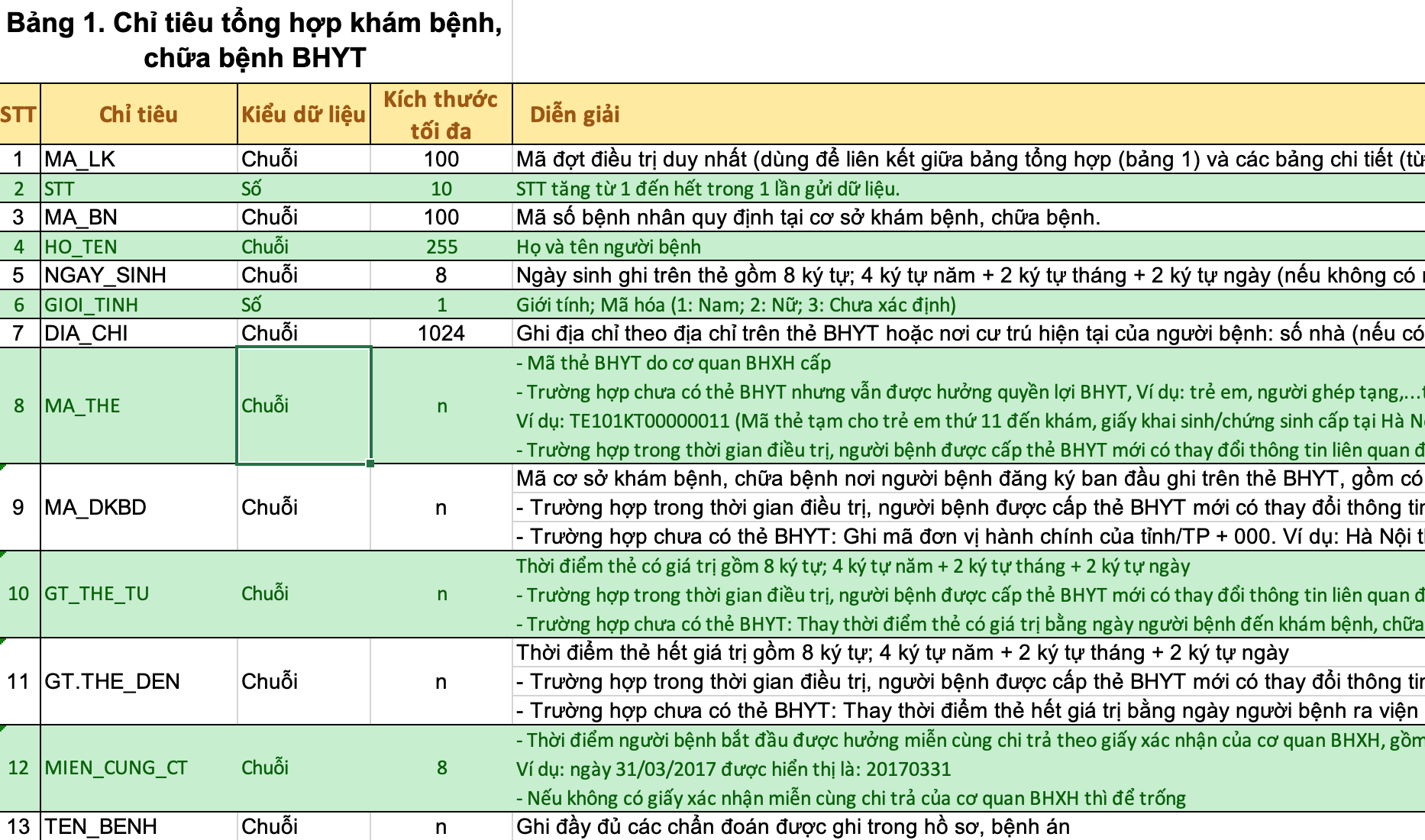
Ở những file trong thư mục con *product*, chứa file *data\_vn*.*xlsx* là file excel để trích xuất dữ liệu. File *data\_eng.xlsx* là file test dựa trên quy chuẩn của [FHIR Specification](https://www.hl7.org/fhir/patient-example.xml.html).

*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.

**

*Hình 2.1.2. Nội dung minh hoạ của file CodeOfConduct.xlsx*

**

*Hình 2.1.3. Nội dung của file data\_vn\_example.xml*

1. **Các thư viện cần có:**

* [openpyxl](https://openpyxl.readthedocs.io/en/stable/): thư viện dùng để đọc/ghi file Excel theo những định dạng xlsx/xlsm/xltx/xltm.
* [yattag](https://www.yattag.org/): thư viện dùng để tạo ra file HTML hoặc XML bằng code Python.
* [datetime](https://docs.python.org/3/library/datetime.html): thư viện để đọc và tính toán ngày tháng trong Python.
* [BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/bs4/doc/): dùng để đọc file xml.
* [queue](https://docs.python.org/3/library/queue.html): dùng để sử dụng *PriorityQueue* nhằm tạo hàng ưu tiên sẽ được đề cập ở phần sau.

1. **Diễn giải code:**
2. **Chuyển từ EXCEL về XML:**
   1. **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]

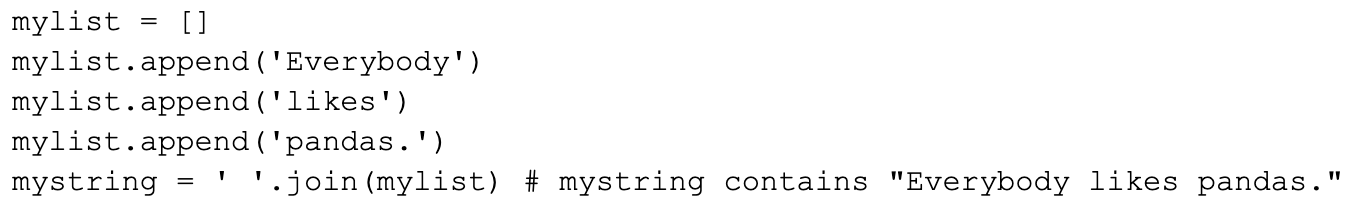
Tiếp đến ta import thư viện *yattag* để xuất định dạng xml:

*from* yattag *import* Doc, indent

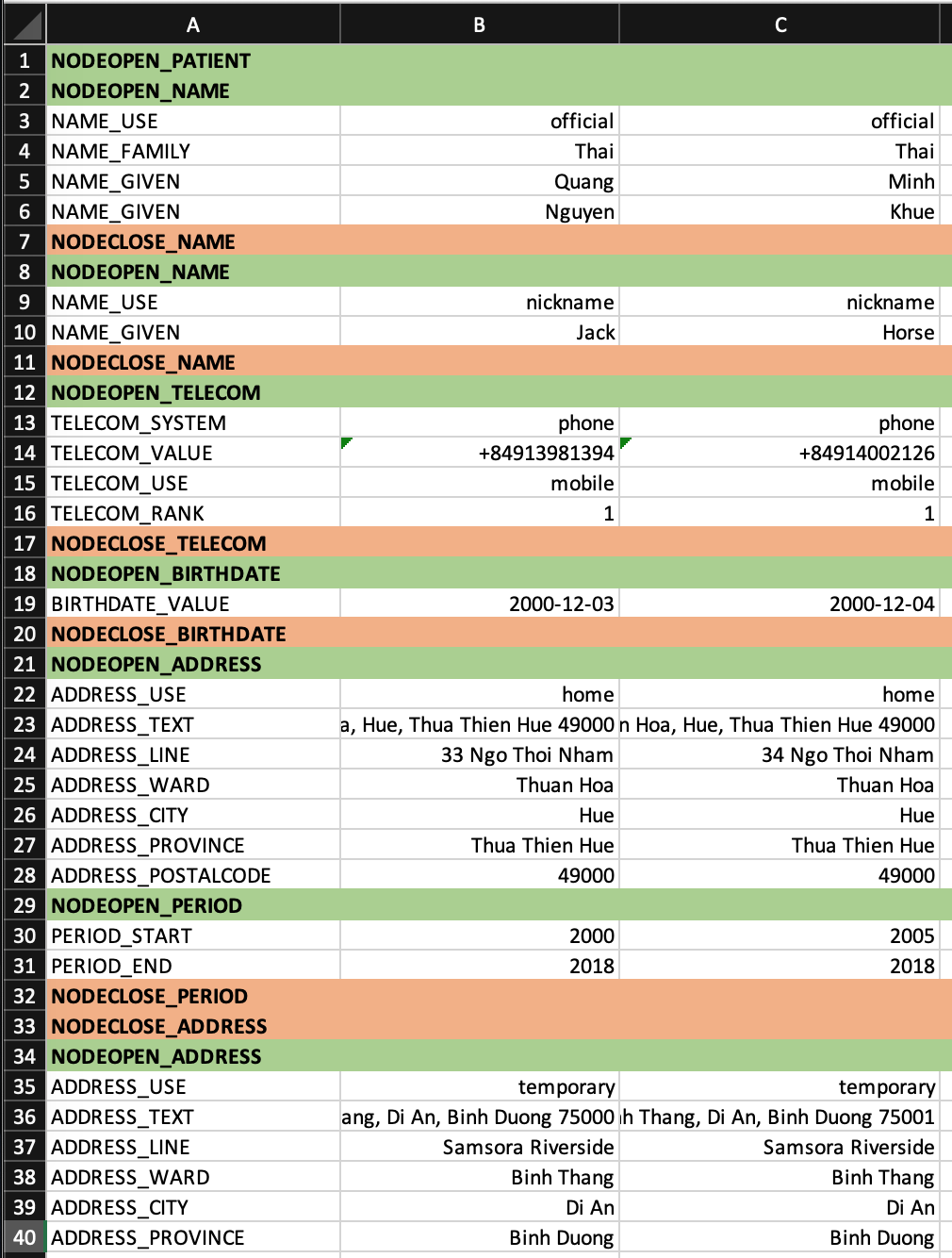
# *Create Yattag doc, tag and text objects*

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:



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

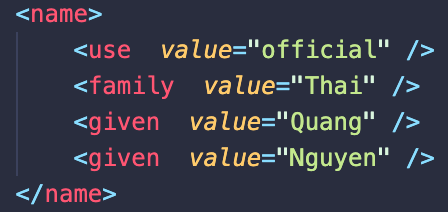


*Hình 3.1.2. Nội dung file data\_vn.xlsx*

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*.



*Hình 3.1.3. Chú thích về sự sắp xếp các tags*

* 1. **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.***

* 1. **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

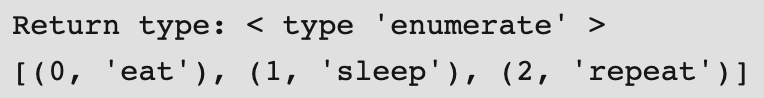
XML\_list.put((\_node.start, \_node.end, \_node.text, \_node))

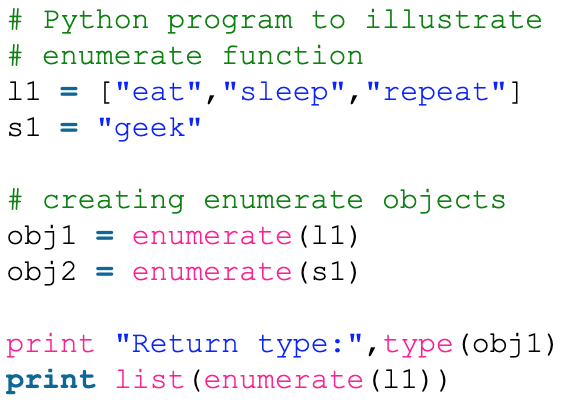
*else*:

XML\_list.put((\_line, -1, \_code[1].lower(), \_data))

*return* \_node

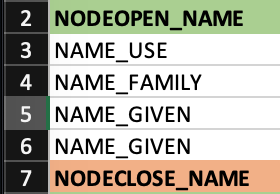
Trong hàm này, ta thực hiện việc quét qua từng hàng của cột A trong file excel (cột chứa tên của các trường và các ký hiệu *NODEOPEN*, *NODECLOSE*). Do chỉ có một cột A nên đối số ***min\_col*** và ***max\_col*** của hàm ***ws.iter\_cols()*** đều bằng 1. Và ***min\_row*** và ***max\_row*** sẽ được gán 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 hoạ như hình dưới:



*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.***

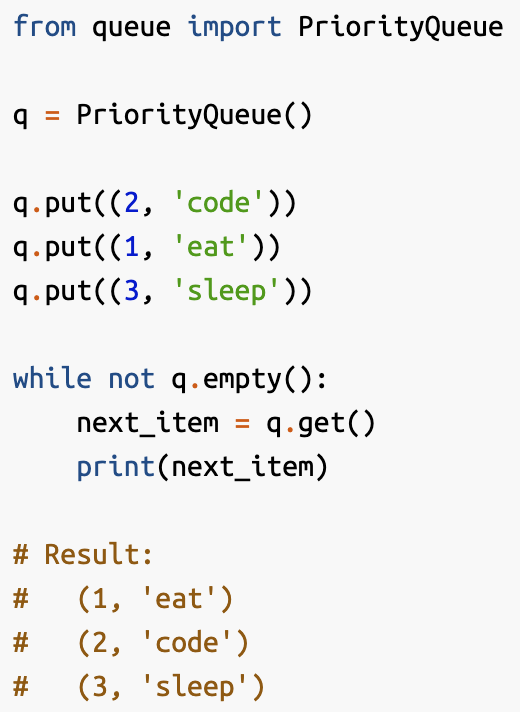
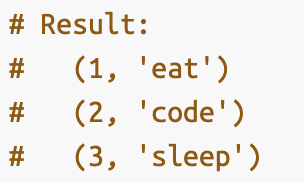
Quay lại với file excel, tác giả đã cố tình lồng tên của tag cha vào điểm mở tag, được ngăn cách bằng ký tự ‘\_’. *NODEOPEN* là lệnh để mở tag, *NAME* chính là tên của tag cần được mở. Chính vì vậy trong hàm *scan\_excelFile()* bên trên, ta ***chia chuỗi này thành 2 chuỗi riêng bằng hàm split()*** và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



*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))

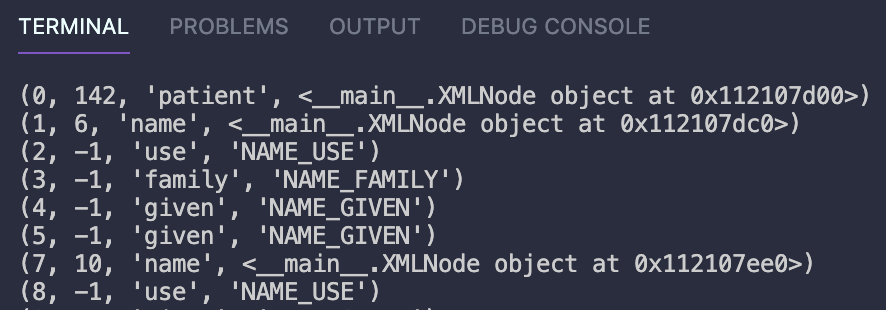
Lưu ý rằng mỗi hàng của XML\_list sẽ có 4 thông số: start, end, text, và object ***XMLNode nếu là tag cha hoặc chuỗi tên hoàn chỉnh của tag trong trường hợp là tag con***. Thông số cuối cùng này để giúp ta phân biệt liệu một tag có phải là object của XMLNode hay không thông qua lệnh ***isinstance()***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

Như vậy sau khi gọi hàm này, ta được một hàng ưu tiên lưu các object XMLNode tượng 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:



*Hình 3.1.6. Kết quả hàm scan\_excelFile()*

* 1. **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]

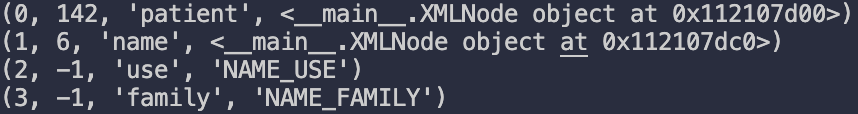
doc.stag(XML\_list2[lineCounter][2], '', value=tmp\_value)

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.

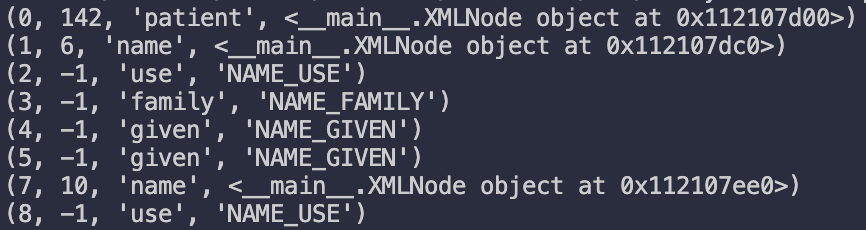


*Hình 3.1.7. Minh hoạ việc đọc file đệ quy*

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 trỏ 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.



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:



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

* 1. **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(),

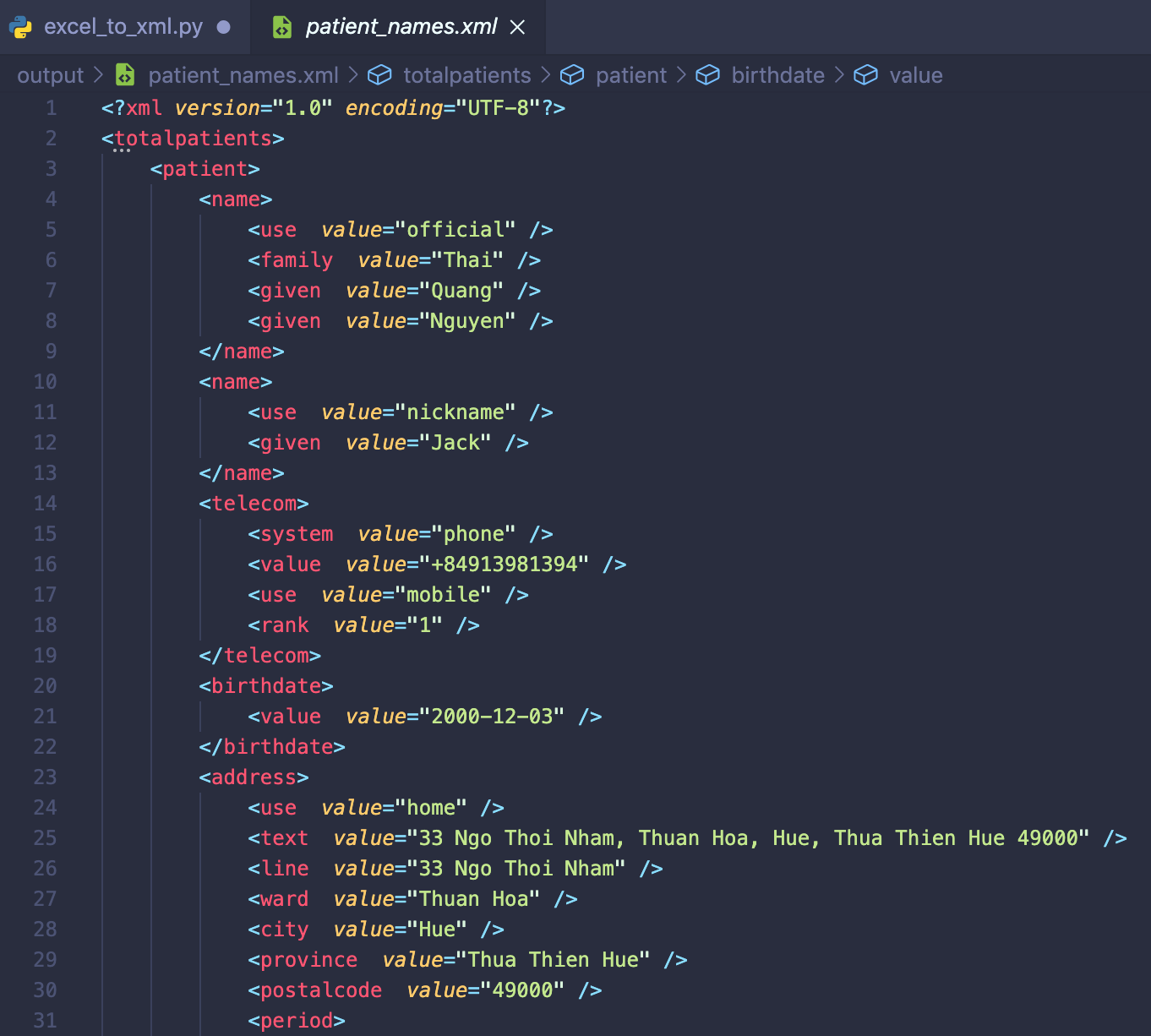
indentation = ' '

)

*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:



*Hình 3.1.8. Dữ liệu lưu trong file patient\_names.xml*

1. **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):

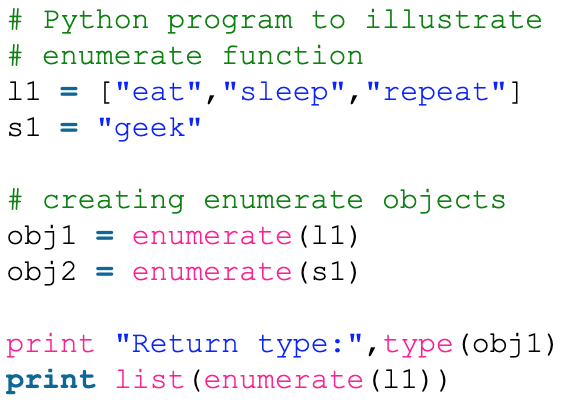
ws.cell(row=i+1, column=j+1).value = value

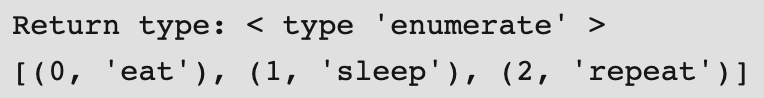
newfilename = os.path.abspath("./output/data\_xml\_to\_excel.xlsx")

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()*:



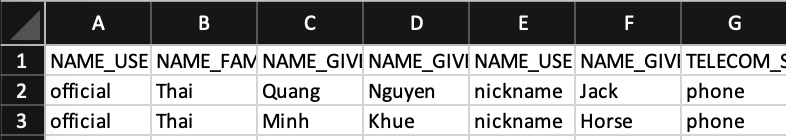
*Hình 3.2.1. Ví dụ về hàm enumerate()*

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)



*Hình 3.2.2. Kết quả file data\_xml\_to\_excel.xlsx*

1. **References:**

<https://stackoverflow.com/>

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

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