**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO

**Bài tập tập quá trình**

**ĐỀ TÀI: ETL Git**

Nhóm 4:

20170057 Nguyễn Thế Đức

20183500 Nguyễn Minh Đức

20183501 Nguyễn Trung Đức

20194027 Lê Tiến Dũng

Giảng viên hướng dẫn: TS. Đỗ Bá Lâm

**Hà Nội, tháng 5 năm 2022**

Mục lục

[**1 Beautiful Soup** 3](#_Toc104757076)

[**1.1** **Giới thiệu** 3](#_Toc104757077)

[**1.2** **Cách cài đặt** 3](#_Toc104757078)

[**1.2.1 Cài đặt beautiful soup** 3](#_Toc104757079)

[**1.2.2 Cài đặt parser** 3](#_Toc104757080)

[**1.3** **Tạo soup** 4](#_Toc104757081)

[**1.3.1** 4](#_Toc104757082)

[**1.3.2** **Điều hướng trong cây** 5](#_Toc104757083)

[**1.3.3** **Tìm kiếm trong cây** 8](#_Toc104757084)

[**2. Blaze** 10](#_Toc104757085)

[**2.1. Giới thiệu** 10](#_Toc104757086)

[**2.2. Nội dung** 10](#_Toc104757087)

[**2.3. Kết luận** 14](#_Toc104757088)

[**3. Bonobo** 16](#_Toc104757089)

[**3.1. Giới thiệu** 16](#_Toc104757090)

[**3.2. Nội dung** 16](#_Toc104757091)

[**3.2.1 Cài đặt bonobo** 16](#_Toc104757092)

[**3.2.2 Các thành phần chính** 16](#_Toc104757093)

[Node 16](#_Toc104757094)

[Graph 16](#_Toc104757095)

[**3.2.3 Tạo một job với bonobo** 17](#_Toc104757096)

[**3.3. Ưu nhược điểm** 20](#_Toc104757097)

[Ưu điểm 20](#_Toc104757098)

[Nhược điểm 20](#_Toc104757099)

[**3.4. Demo** 20](#_Toc104757100)

# **1 Beautiful Soup**

## **Giới thiệu**

BeautifulSoup là một thư viện Python dùng để lấy dữ liệu ra khỏi các file HTML và XML. Nó hoạt động cùng với các parser (trình phân tích cú pháp) cung cấp cho bạn các cách để điều hướng, tìm kiếm và chỉnh sửa trong parse tree (cây phân tích được tạo từ parser). Nhờ các parser này nó đã giúp các lập trình viên tiết kiệm được nhiều giờ làm việc.

## **Cách cài đặt**

### **1.2.1 Cài đặt beautiful soup**

* Nếu sử dụng một phiên bản Debian hay Ubuntu Linux, có thể cài đặt Beautiful Soup bằng system package manager:

apt-get install python-bs4 (Python 2)

apt-get install python3-bs4 (Python 3)

* Thông qua pip hoặc easy\_install: (sử dụng đúng phiên bản pip hoặc easy\_install tương ứng với phiên bản python tên có thể là pip3 hoặc easy\_install3 với python3)

pip install beautifulsoup4

easy\_install beautifulsoup4

### **1.2.2 Cài đặt parser**

Beautiful Soup hỗ trợ parser HTML có trong thư viện chuẩn của Python, nhưng nó cũng hỗ trợ một số parser Python của bên thứ ba.

* Lxml

apt-get install python-lxml

easy\_install lxml

pip install lxml

* html5lib parser

apt-get install python-html5lib

easy\_install html5lib

pip install html5lib

## **Tạo soup**

Để bóc tách một tài liệu chỉ cần truyền nó vào constructor của BeautifulSoup dưới dạng string hoặc File object đang mở, hàm sẽ trả về một BeautifulSoup object biểu diễn cây phân tích được.

**from** **bs4** **import** BeautifulSoup

**with** open("index.html") **as** fp:

soup = BeautifulSoup(fp)

soup = BeautifulSoup("<html>data</html>")

Đầu tiên tài liệu được chuyển đổi thành Unicode, và các phần tử HTML được chuyển đổi sang kí tự Unicode. Sau đó nó sẽ được bóc tách bằng parser tốt nhất hiện có.

### 

BeautifulSoup sẽ biến một tài liệu HTML phức tạp thành một cây object mà ở đó ta chủ yếu chỉ thao tác với các object bao gồm: Tag, NavigableString, BeautifulSoup, Comment.

#### Tag

Một Tag object tương ứng với một tag (còn được gọi là một thẻ hoặc một phần tử - element) trong tài liệu html hay xml.

VD: div, span, p, h1, a, ...

<b id=”greeting”>Hello</b>

Một Tag có chứa nhiều thuộc tính và phương thức.Trong đó quan trọng nhất là Name và Attributes

* Name: Tag nào cũng có tên, ta có thể lấy nó ra thông qua thuộc tính .name
* Attributes:
  + Một tag có thể chứa nhiều thuộc tính, có thể truy cập nó như 1 dict của Python:
    - VD: b[“id]
  + Hoặc thông qua thuộc tính .attrs
    - VD b.attrs sẽ trả về dict chứa các thuộc tính của tag b
  + Các thuộc tính có nhiều giá trị phổ biến nhất là class ngoài ra có rel, rev, accept-charset, headers và accesskey. Beautiful Soup sẽ trả về 1 list biểu diễn các giá trị của thuộc tính đó

#### 1.3.1.2 NavigableString

Một chuỗi tương ứng với một văn bản chứa trong một tag. Beautiful Soup sử dụng class NavigableString để chứa những văn bản này. Truy cập thông qua thuộc tính .string của Tag.

Một NavigableString giống như một chuỗi Unicode trong Python, ngoài ra nó còn được hỗ trợ thêm một vài tính năng như điều hướng trong cây và tìm kiếm trong cây, có thể chuyển đổi một NavigableString thành một chuỗi Unicode bằng hàm str().

#### BeautifulSoup

Bản thân BeautifulSoup object đại diện cho toàn bộ tài liệu. Nhưng với hầu hết các mục đích sử dụng, có thể sử dụng nó như một Tag object.

Vì đối tượng BeautifulSoup không phải là một tag HTML hay XML thực tế nào, nên nó không có name và attribute. Nhưng khi truy cập .name của nó, nó sẽ trả về một .name đặc biệt đó là “[document]”.

#### Coment và các chuỗi đặc biệt khác

Comment object là một dạng đặc biệt của NavigableString.

<b><!--Hey, buddy. Want to buy a used parser?--></b>

Ngoài ra, Beautiful Soup định nghĩa các class cho bất cứ thứ gì khác có thể xuất hiện trong một tài liệu XML: Cdata, ProcessingInstruction, Declaration, và Doctype. Như Comment, tất cả các class này đều là class con của NavigableString, nó sẽ thêm một cái gì đó vào chuỗi.

### **Điều hướng trong cây**

Các Tag có thể chứa string hoặc một tag khác. Những phần tử như này gọi là tag con (tag's children). BeautifulSoup cung cấp rất nhiều thuộc tính khác nhau để điều hướng và lặp qua các tag con này.

Lưu ý: các Beautiful Soup string không hỗ trợ các thuộc tính này vì một string không thể có tag con.

#### 1.3.2.1 Điều hướng bằng tên thẻ

Cách đơn giản nhất để điều hướng trong một parse tree là sử dụng tên của tag mà bạn muốn duyệt. Nếu bạn muốn duyệt tag <head>, hãy dùng soup.head.

Bạn có thể sử dụng cách này nhiều lần để đi xuống một phần nhất định nằm sâu trong parse tree.

VD soup.body.b sẽ lấy ra được thẻ <b> bên dưới thẻ <body>

Sử dụng tên thẻ như một thuộc tính ta sẽ chỉ lấy được thẻ đầu tiên bằng tên đó. Nếu bạn cần lấy hết tất cả các thẻ bạn sẽ cần một phương thức của tìm kiếm là find\_all.

VD soup.find\_all(‘a’)

#### 1.3.2.2 Sử dụng .content và .children

Các tag con được chứa trong một list được gọi là .contents.

Bản thân BeautifulSoup object cũng có một thẻ con. Trong trường hợp này, thẻ <html> là thẻ con của BeautifulSoup object.

soup.contents[0].name

# u'html'

Một string thì không có .content, bởi vì nó không chứa thứ gì.

Thay vì truy cập thẻ con trong một list, ta có thể lặp qua một thẻ con bằng cách sử dụng .children generator

**for** child **in** title\_tag.children:

    print(child)

#### Sử dụng .descendants

Hai thuộc tính .contents và .children chỉ duyệt qua một phần tử con trực tiếp của thẻ. Để truy cập tất cả các phần tử con cả trực tiếp và gián tiếp cần sử dụng .descendants, .descendants cho bạn lặp qua các thẻ con, bằng cách đệ quy.

**for** child **in** head\_tag.descendants:

    print(child)

#### Sử dụng .string

Nếu một thẻ chỉ có một phần tử con, và nó là một NavigableString, thì phần tử con đó sẽ được gọi ra bằng .string

title\_tag.string

# u'The Dormouse's story'

Nếu phần tử con của một thẻ là một thẻ khác và thẻ đó có một .string, thì thẻ cha được coi là có cùng .string với thẻ con của nó.

head\_tag.contents

# [<title>The Dormouse's story</title>]

head\_tag.string

# u'The Dormouse's story'

Nếu một thẻ chứa nhiều hơn một phần tử con, thì nó sẽ không hiểu .string nói đến cái gì. Vì vậy .string được định nghĩa là None

#### Sử dụng .strings and .stripped\_strings

Nếu có nhiều hơn một phần tử con có trong một thẻ. Bạn cũng có thể lấy ra những string có trong những thẻ đó. Bằng cách sử dụng bộ tạo .strings:

**for** string **in** soup.strings:

    print(repr(string))

# u"The Dormouse's story"

# u'\n\n'

# u"The Dormouse's story"

# u'\n\n'

# u'Once upon a time there were three little sisters; and their names were\n'

# u'Elsie'

# u',\n'

# u'Lacie'

# u' and\n'

# u'Tillie'

# u';\nand they lived at the bottom of a well.'

# u'\n\n'

# u'...'

# u'\n'

Những chuỗi này có xu hướng có thêm những khoảng trắng bổ xung, bạn có thể loại bỏ chúng bằng cách sử dụng bộ tạo .stripped\_strings, các chuỗi chứa toàn khoảng trắng sẽ bị bỏ qua, và khoảng trắng ở phần bắt đầu và kết thúc của một chuỗi cũng sẽ bị loại bỏ.

**for** string **in** soup.stripped\_strings:

    print(repr(string))

# u"The Dormouse's story"

# u"The Dormouse's story"

# u'Once upon a time there were three little sisters; and their names were'

# u'Elsie'

# u','

# u'Lacie'

# u'and'

# u'Tillie'

# u';\nand they lived at the bottom of a well.'

# u'...'

Ngoài ra còn nhiều phương thức điều hướng khác đi lên trên parent hoặc đi ngang giữa các phần tử cùng cấp.

### **Tìm kiếm trong cây**

Beautiful Soup định nghĩa rất nhiều phương thức dùng để tìm kiếm trong parse tree, nhưng chúng rất giống nhau, hai phương thức phổ biến nhất: find() và find\_all()

Phương thức find\_all() sẽ duyệt qua tất cả các thẻ con và lấy tất cả các thẻ mà phù hợp với bộ lọc.

Phương thức find\_all() quét toàn bộ tài liệu để tìm kiếm kết quả. Nhưng khi chỉ muốn tìm một kết quả ta thể sử dụng phương thức find()

#### 1.3.3.1 Các loại bộ lọc

String: Đây là bộ lọc đơn giản nhất. Truyền một chuỗi vào một phương thức tìm kiếm và BeautifulSoup sẽ thực hiện so khớp chính xác với chuỗi đó.

VD code tìm tất cả thẻ <b> có trong tài liệu:

soup.find\_all('b')

Regular Expression: Nếu truyền vào một regular expression object, Beautiful Soup sẽ lọc theo regular expression đó bằng cách sử dụng phương thức search() của nó.

VD code tìm tất cả các thẻ có tên bắt đầu bằng chữ “b” chẳng hạn <b> và thẻ <body>

**import** re

**for** tag **in** soup.find\_all(re.compile("^b")):

    print(tag.name)

List: Nếu truyền vào một list, Beautiful Soup sẽ cho phép một chuỗi match bất kì phần tử nào có trong list đó.

VD Code tìm tất cả thẻ <a> và thẻ <b>:

soup.find\_all(["a", "b"])

True: Giá trị True sẽ so khớp tất cả.

VD code tìm tất cả thẻ có trong tài liệu, nhưng không có text string nào:

**for** tag **in** soup.find\_all(**True**):

    print(tag.name)

Function: Nếu không có matcher nào phù hợp, có thể tự định nghĩa một function có 1 param là tag (param này mặc định sẽ được truyền vào một element). Hàm này trả về True nếu element đó match, và ngược lại trả về False.

VD:

**def** **has\_class\_but\_no\_id**(tag):

**return** tag.has\_attr('class') **and** **not** tag.has\_attr('id')

soup.find\_all(has\_class\_but\_no\_id)

# **2. Blaze**

## **2.1. Giới thiệu**

Blaze dịch một tập hợp con của cú pháp giống NumPy và Pandas đã được sửa đổi sang cơ sở dữ liệu và các hệ thống máy tính khác. Blaze cho phép người dùng Python một giao diện quen thuộc để truy vấn dữ liệu ở trong các hệ thống lưu trữ dữ liệu khác.

Blaze không thực hiện tính toán. Nó dựa vào các hệ thống khác như SQL, Spark hoặc Pandas để thực hiện việc xử lý số thực. Nó không phải là sự thay thế cho bất kỳ hệ thống nào trong số này.

Blaze không triển khai toàn bộ API NumPy / Pandas, cũng như không tương tác với các thư viện làm việc với NumPy / Pandas.

Blaze là một cách tốt để kiểm tra dữ liệu ở trong cơ sở dữ liệu lớn, thực hiện một tập hợp nhỏ nhưng mạnh mẽ để truy vấn dữ liệu đó và sau đó chuyển đổi kết quả của bạn thành định dạng phù hợp với các công cụ Python quen thuộc.

Một số dự án cung cấp phân tích dữ liệu phong phú và hiệu quả. Sự cạnh tranh giữa các dự án này làm phát sinh một hệ sinh thái sôi động và năng động. Blaze tăng cường hệ sinh thái này với giao diện đồng nhất và có thể thích ứng. Blaze sắp xếp việc tính toán và truy cập dữ liệu giữa các dự án bên ngoài này. Nó cung cấp một bối cảnh nhất quán để xây dựng các giao diện tiêu chuẩn có sẵn bởi cộng đồng Python hiện tại.

## **2.2. Nội dung**

**\* Cách cài đặt**

Blaze có thể được cài đặt dễ dàng nhất từ [​​conda](http://conda.pydata.org/):

$ conda install blaze

Các bản dựng cập nhật khác có sẵn trên kênh anaconda: <http://anaconda.org/blaze>

conda install -c blaze blaze

Blaze cũng có thể được cài đặt bằng cách sử dụng pip:

pip install blaze --upgrade

**or**

pip install git+https://github.com/blaze/blaze --upgrade

Nếu bạn quan tâm đến phiên bản phát triển của Blaze, bạn có thể lấy nguồn từ Github.

$ git clone git@github.com:blaze/blaze.git

Blaze phụ thuộc vào NumPy, Pandas và một vài thư viện thuần python. Nó sẽ dễ dàng cài đặt trên bất kỳ thiết lập Numeric Python nào.

[numpy](http://www.numpy.org/) > = 1,7

[datashape](https://github.com/blaze/datashape) > = 0,4,4

[odo](https://github.com/blaze/odo) > = 0,3,1

[toolz](http://toolz.readthedocs.org/) > = 0,7.0

[cytoolz](https://github.com/pytoolz/cytoolz/)

[multipledispatch](http://multiple-dispatch.readthedocs.org/)> = 0,4,7

[pandas](http://pandas.pydata.org/)

**\* Trong quá trình cài đặt em có gặp 1 vài lỗi về odo có thể do version đã lâu không cập nhật:**

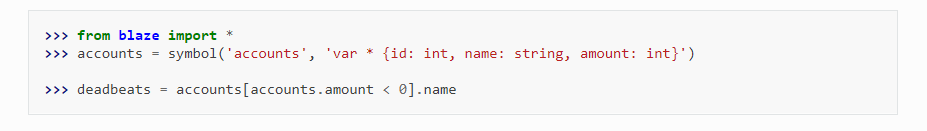
**Có thể fix bằng cách như sau:**

[**AttributeError: module 'pandas' has no attribute 'tslib' when I use blaze module**](https://stackoverflow.com/questions/64326138/attributeerror-module-pandas-has-no-attribute-tslib-when-i-use-blaze-module)

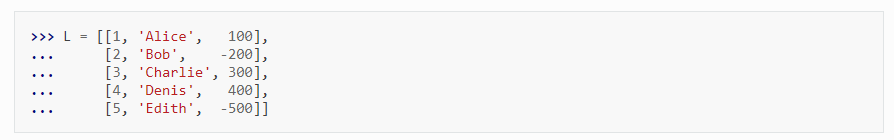
<https://stackoverflow.com/questions/64326138/attributeerror-module-pandas-has-no-attribute-tslib-when-i-use-blaze-module>

**\* Cách hoạt động của Blaze**

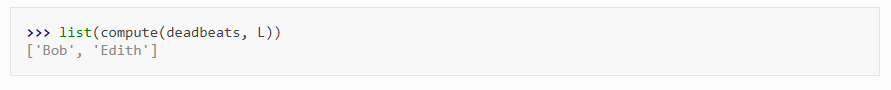
Blaze phân tách các phép tính mà chúng ta muốn thực hiện:



Từ việc biểu diễn dữ liệu:



Blaze cho phép người dùng giải quyết các vấn đề theo hướng dữ liệu:



Nhưng việc tách biểu thức khỏi dữ liệu cho phép chúng ta chuyển đổi giữa các phần phụ trợ khác nhau. Ở đây Blaze giải quyết vấn đề tương tự bằng cách sử dụng Pandas thay vì Pure Python.

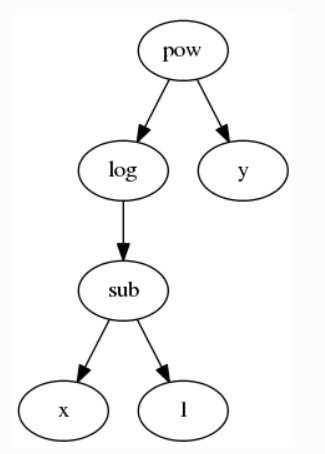


Blaze không tính toán những kết quả này, thay vào đó Blaze thúc đẩy các dự án khác tính toán chúng một cách thông minh. Các dự án này bao gồm từ các trình vòng lặp Pure Python đơn giản đến các cụm Spark được phân phối mạnh mẽ. Blaze được xây dựng để mở rộng cho các hệ thống mới khi chúng phát triển.

**\*Thiết kế Expression**

Về cốt lõi, Blaze thể hiện các truy vấn phân tích một cách tượng trưng. Nó biểu diễn các truy vấn này như một cây biểu thức trừu tượng. Cây này ra lệnh cho người dùng tương tác, tối ưu hóa, và cuối cùng được chuyển sang các hệ thống tính toán khác. Không cần thiết phải có kiến ​​thức sâu về các biểu thức trừu tượng để sử dụng Blaze; nó là điều cần thiết để phát triển trong đó.

Hãy xem xét ví dụ sau:

****

**>>> from** **blaze** **import** symbol, log

**>>>** x = symbol('x', 'int64')

**>>>** y = symbol('y', 'float32')

**>>>** z = log(x - 1)\*\*y

Chúng ta biểu diễn biểu thức toán học dưới dạng một cây trong đó mọi toán tử (ví dụ , ) là một nút mà các nút con của nó là đối số của nó. Tất cả các biểu thức Blaze và thực sự là tất cả các biểu thức trong bất kỳ ngôn ngữ lập trình nào đều có thể được in lại theo cách này. Blaze mã hóa cây này như một cấu trúc dữ liệu trong đó mọi nút là một đối tượng có kiểu tương ứng với hoạt động của nó. Ví dụ, tồn tại các lớp sau:

**class** **pow**(Expr):

...

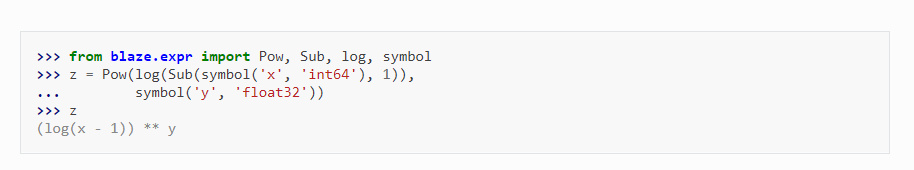
**class** **sub**(Expr):

...

**class** **log**(Expr):

...

Blaze thấy, sẽ giống như sau:

****

## **2.3. Kết luận**

Blaze giao tiếp với Python và Pandas như đã thấy ở trên và một số công nghệ khác, bao gồm NumPy, SQL, Mongo, Spark, PyTables, v.v. Blaze được xây dựng để giúp kết nối với công nghệ mới dễ dàng.

Blaze hiện nhắm mục tiêu vào cơ sở dữ liệu và công nghệ được sử dụng cho các truy vấn phân tích. Nó cố gắng sắp xếp và cung cấp các giao diện trên cùng và giữa các hệ thống tính toán khác. Chúng tôi cung cấp hiệu suất bằng cách cung cấp cho các nhà khoa học dữ liệu quyền truy cập trực quan vào nhiều công cụ khác nhau.

Blaze đôi khi bị cường điệu quá mức. Thuật ngữ Bigdata và Pandas chắc chắn khiến tâm trí mọi người trở thành một thứ gì đó không thể đạt được và dẫn đến thất vọng. Blaze có giới hạn; học những hạn chế đó có thể hướng bạn đến năng suất cao hơn.

Đầu tiên và quan trọng nhất, Blaze không thay thế Pandas. Pandas sẽ luôn có nhiều tính năng phong phú hơn và trưởng thành hơn Blaze.  Nếu dữ liệu của bạn vừa khít trong bộ nhớ thì hãy sử dụng NumPy / Pandas.

**\*Một số điều cụ thể Blaze không làm:**

1. Làm sạch dữ liệu phi cấu trúc. Blaze chỉ xử lý các truy vấn phân tích trên dữ liệu có cấu trúc.
2. Hầu hết mọi thứ trong SciPy. Bao gồm những thứ như FFT và gradient descent.
3. Hầu hết mọi thứ trong SciKit Learn / Image / etc ..
4. Suy luận thống kê - Chúng tôi mời bạn xây dựng cái này (cái này thực sự khá khả thi.)
5. Song song mã Python hiện có của bạn
6. Thay thế Spark - Blaze có thể hoạt động trên Spark, nó không cạnh tranh với nó.
7. Tính toán nhanh chóng - Blaze sử dụng những thứ khác để tính toán, bản thân nó không tính toán bất cứ thứ gì. Vì vậy, việc đặt câu hỏi về tốc độ của Blaze được quyết định hoàn toàn bởi tốc độ của những thứ khác.

**\*Blaze làm gì:**

Blaze là một hệ thống truy vấn trông giống như NumPy / Pandas. Bạn viết các truy vấn Blaze, Blaze dịch các truy vấn đó sang một thứ khác (như SQL) và chuyển các truy vấn đó đến cơ sở dữ liệu khác nhau để chạy trên mã nhanh của người khác. Nó làm trơn tru quá trình này để làm cho việc tương tác với dữ liệu nước ngoài có thể truy cập được như sử dụng Pandas. Điều này thực sự khá khó khăn.

Blaze tăng khả năng tiếp cận của con người chứ không phải hiệu suất tính toán.

# **3. Bonobo**

## **3.1. Giới thiệu**

Bonobo là một framwork, một thư viện ETL (Etract- Tranform- Load) khá nhẹ được viết bằng Python, hỗ trợ Python 3.5+.

Nó cung cấp công cụ cho việc xây dựng các đường ống (pipelines) để chuyển đổi dữ liệu, bằng cách sử dụng python thuần, và thực hiện nó một cách song song

Nó hoạt động bằng cách truyền dữ liệu trực tiếp thông qua một đồ thị có hướng của các hàm python, dữ liệu được truyền vào row(có thể là một object hoặc một bản ghi SQL) theo kiểu hàng đợi (queue) first in first out

Bonobo có thể làm việc với các định dạng dữ liệu phổ biến (XML, JSON, CSV, EXCEL,...) và các dịch vụ phổ biến (các cơ sở dữ liệu SQL hay REST web service)

## **3.2. Nội dung**

### **3.2.1 Cài đặt bonobo**

Yêu cầu:

* Python 3.5 +
* 

Nên sử dụng python venv để tránh xung đột giữa các thư viện liên quan

### **3.2.2 Các thành phần chính**

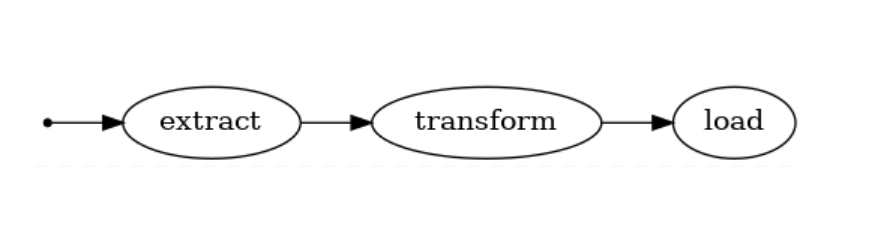
### Node

Trên các Graph ta định nghĩa các Node để xử lý dữ liệu mỗi Node trên graph đều có input và output, output của Node này có thể là input của Node khác.

Mỗi Node có thể sinh ra ouput chuyển đến hoặc nhận input từ nhiều Node khác

### Graph

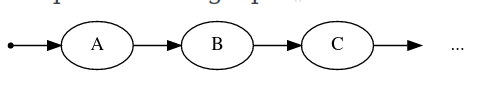
Graph trong bonobo được định nghĩa là một đồ thị có hướng đại diện cho đường ống để thực hiện các công việc để chuyển đổi dữ liệu. Mỗi graph sẽ đại diện cho job mà ta thực hiện. Dưới đây là một ví dụ:



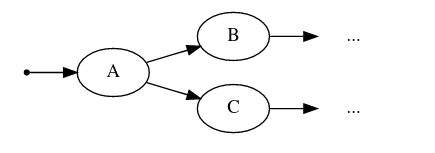
Dữ liệu sẽ được truyền vào Graph, xử lý và sinh ra output

Các loại Graph cơ bản:

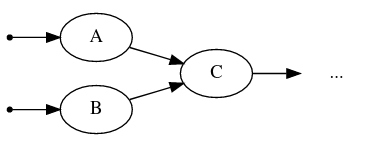
* Linear Graph



* Graph có điểm phân kỳ



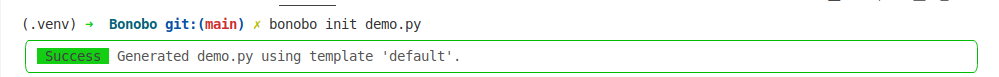
* Graph có điểm hội tụ

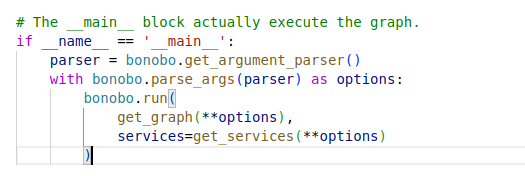


### **3.2.3 Tạo một job với bonobo**

* Khởi tạo job

Sử dụng câu lện bonobo init

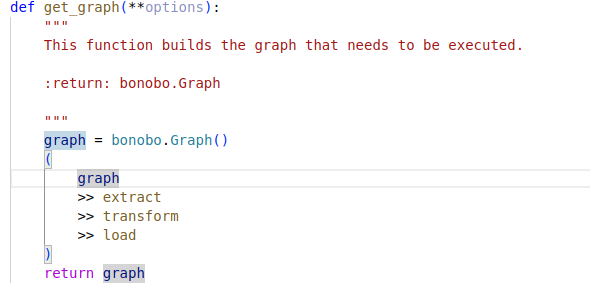
Sau khi thành công sẽ trả về thông báo như trên.



Bonobo sẽ tự động sinh một file demo.py với hàm main như trên. Đoạn mã trên thể hiện luồng chính của job được khởi tạo. Ở đây chúng ta sẽ quan tâm đến một số hàm chính như bonobo.run(), get\_graph() và get\_services()

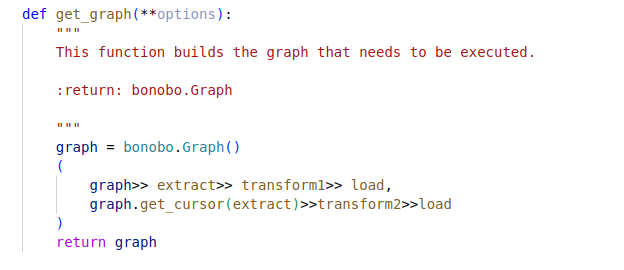
* Định hình Graph

Để tạo một job phục vụ việc extract, transform và load dữ liệu, việc đầu tiền là ta cần định nghĩa một đường ống để luân chuyển dữ liệu, tức là xây dựng một đồ thị (Graph) có hướng theo cú pháp của bonobo.



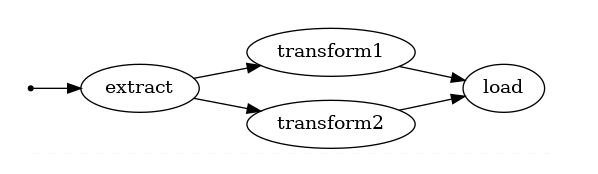
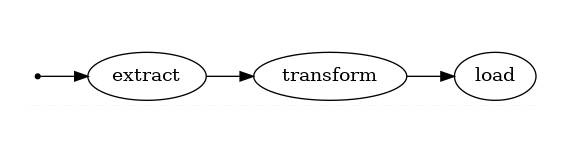
Đoạn mã bên trên xây dựng một Graph cơ bản gồm ba Node : extract, transform, load, trả về một bonobo.Graph. Mỗi một Node đảm nhận một công việc nào đó trong đường ống.

Ta có thể tùy chỉnh graph để phù hợp vợi job mà ta thực hiện



Ta có thể export Graph dưới dạng hình ảnh với câu lệnh sau:

Sau đây là 2 biểu đồ được thể hiện bởi hai đoạn mã trên:



Sau khi đã xây dựng được một Graph thể hiện cho đường ống chuyển đổi dữ liệu, công việc tiếp theo là định nghĩa chi tiết các Node

* Xây dựng các Node



Đoạn mã trên xây dựng các Node cơ bản cho ví dụ Graph ở trên. Dữ liệu sẽ được xử lý song song giữa các Node và theo thứ tự hàng đợi tại mỗi Node

## **3.3. Ưu nhược điểm**

### Ưu điểm

* Cấu trúc của Bonobo rất trực quan, giúp người đọc có cái nhìn rõ ràng về quy trình xử lý dữ liệu
* Bonobo xây dựng hướng đến việc chuyển đổi dữ liệu ở thế giới thực, vì vậy nó có thể hỗ trợ việc xây dựng mọi luồng dữ liệu
* Bonobo chỉ sử dụng Python thuần
* Bonobo là một thư viện mã nguồn mở

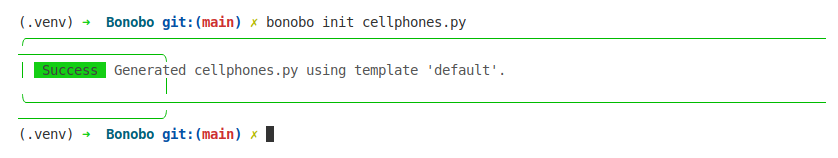
### Nhược điểm

* Bonobo chỉ hướng đến cung cấp cấu trúc để thực thi luồng dữ liệu
* Các công cụ xử lý tại các Node như praser vẫn còn chưa hỗ trợ => Việc xử lý dữ liệu tại các Node hầu như phải sử dụng đến các thư viện hoặc extention khác
* Việc cài đặt còn nhiều lỗi do xung đột phiên bản giữa các thư viện Python phụ thuộc

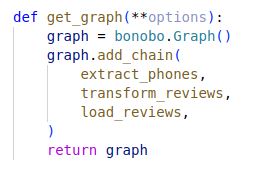
## **3.4. Demo**

Trong ví dụ này, chúng mình sẽ tạo một job cơ bản với bonobo để thực hiện việc Etract, Tranform và Load dữ liệu là các sản phẩm điện thoại trên trang cellphones.com.vn

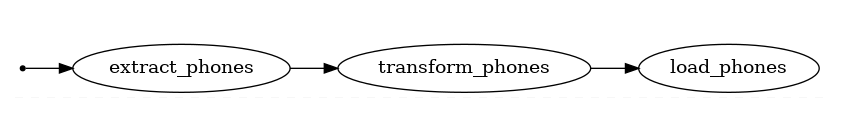
* Đầu tiên khởi tạo job



* Tiếp theo tại hàm get\_graph(), ta xây dựng một Graph cho luồng dữ liệu, giả sử có 3 nút như sau:

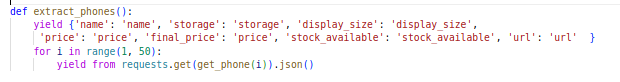


Ta có thể xem Graph dưới dạng hình vẽ bằng câu lệnh:



* Sau khi có Graph đại diện cho luồng dữ liệu, tiếp theo ta xây dựng chi tiết từng Node bên trong
  + Etract\_phones





Ở đây chúng mình sử dụng api tìm kiếm được để request lên server của cellphones, api được lấy bằng hàm get\_phone() với tham số page là số trang, mỗi vòng lăp sẽ trả về dữ liệu là một danh sách điện thoại dứoi dạng object json được convert thành một Python Tupple và chuyển đến Node tiếp theo. (Mỗi phần tử trong tupple được tính là 1 row theo thuật ngữ của Bonobo)

Note: Mặc dù trả về một danh sách, Tuy nhiên danh sách đó sẽ chỉ được thêm vào hàng đợi. Các object sẽ được xử lý độc lập với nhau

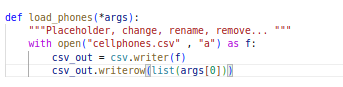
* + Transform\_phones



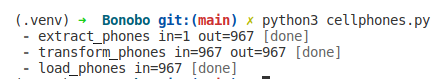
Với đầu vào là một row dữ liệu, thực hiện việc chuyển đổi dữ liệu, ở đay là chọn lựa các trường trong object extract được từ bước trước.

* + Load\_phones

Lưu trữ dữ liệu vào file dạng csv



* Sau khi đã có Graph và các nút hoàn chỉnh, ở terminal chạy câu lệnh thực thi job:



Kết quả chạy được thể hiện như hình. Mở file dữ liệu thu thập bằng LiBreOffice (Linux) có thể thấy kết quả như sau:

