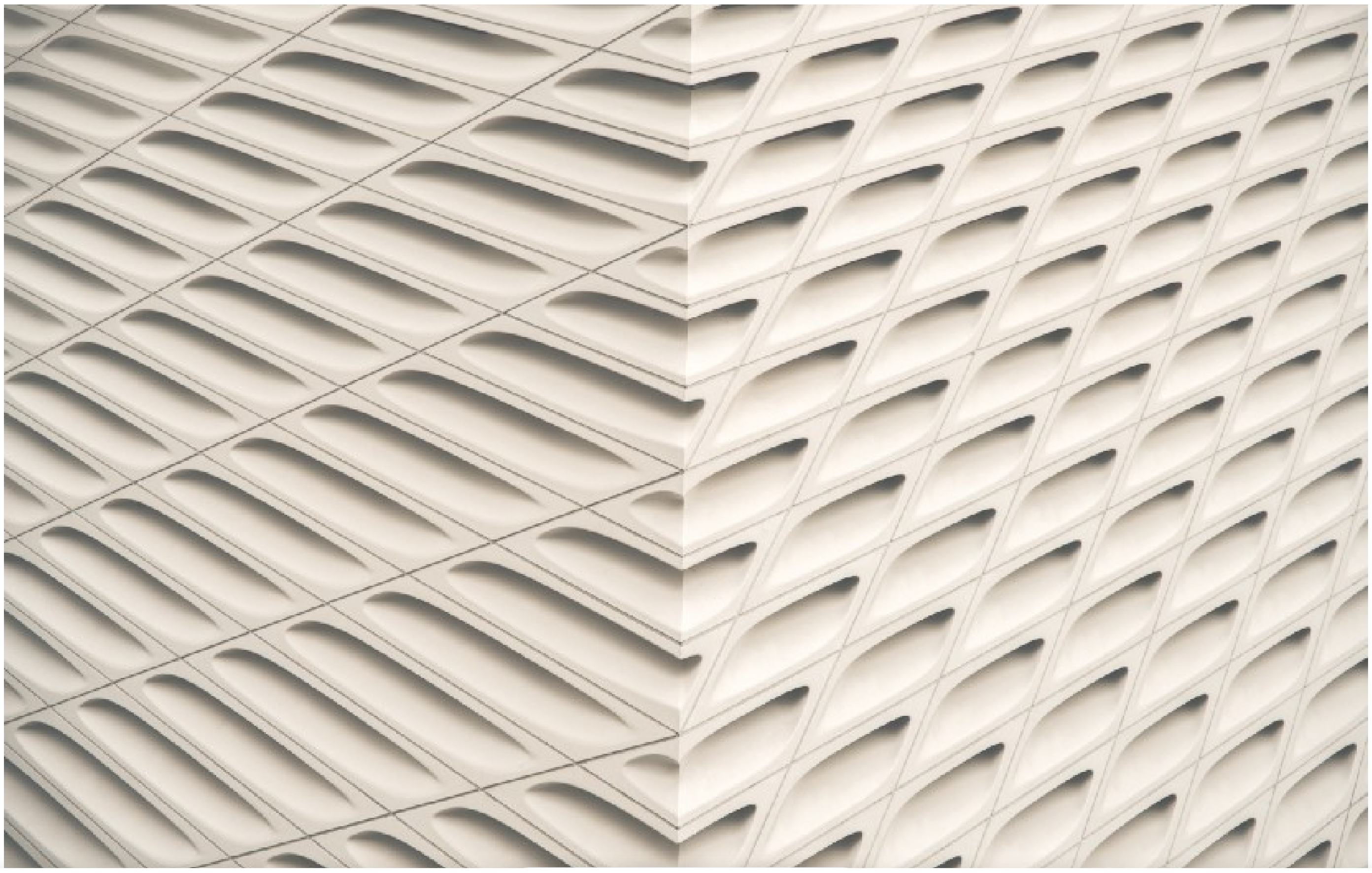
Daniel Ellis Research Follow
Apr 12 · 5 min read · · Listen

Save

Language Translation Using Python

How to translate a set of prose, and/or XML pages using a number of language-translation APIs.



Although online translation tools are available, it is not always possible to access them, or it may be better to provide a static translated page for our clients.

In this tutorial, we look at the different translation APIs available, how to use them in Python, and how we can use BeutifulSoup to translate text within HTML webpages or XML documents.

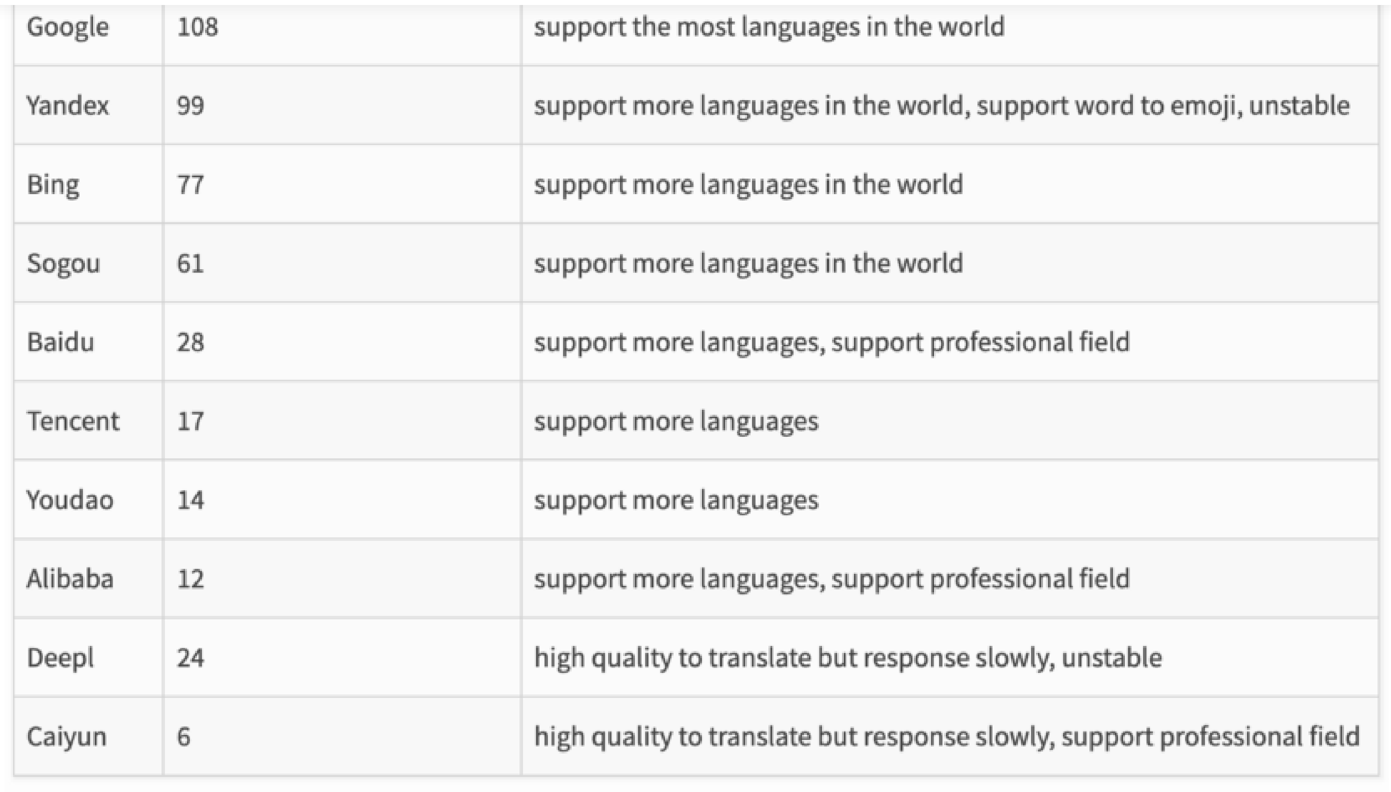
Installing the Library

To do our translation we use the `translate-api` python library. This provides an effective interface to the many possible translation APIs that are available.

```
pip install translators --upgrade
```

Selecting Translator

We begin by selecting a translator. Although Google Translate is often our default there are several more precise ones that are available:



Google	108	support the most languages in the world
Yandex	99	support more languages in the world, support word to emoji, unstable
Bing	77	support more languages in the world
Sogou	61	support more languages in the world
Baidu	28	support more languages, support professional field
Tencent	17	support more languages
Youdao	14	support more languages
Alibaba	12	support more languages, support professional field
Deepl	24	high quality to translate but response slowly, unstable
Caiyun	6	high quality to translate but response slowly, support professional field

Source: https://pypi.org/project/translate-api/

It is worth noting that more specific translators also support a professional field for their translations — this ensures that the translation matches those which have been derived from documents (a corpus) within the specified field.

Two examples are given below and provided using the 'professional_field' argument when specifying a query.

```
baidu: ('common','medicine','electronics','mechanics')
caiyun: ("medicine","law","machinery")
```

Constructing a Query

Now we have chosen a translation API we load our script and address it as follows.

There are a number of languages between which we may translate. To check if the two we are interested in, we can select our translator and run the following command:

Note the underscore before the translator name!

```
ts._<translator_name>.language_map # e.g. ts._google.language_map
```

This gives us a dictionary of keys for each included language, and values representing what they may be translated into.

In the case of Google, we can also look at the Language Support page below:

Language support | Cloud Translation | Google Cloud
Send feedback The Translation API's recognition engine supports a wide variety of languages for the Neural Machine…
cloud.google.com

Viewing the Options

If we decide to use the Google API for our translation, we can use the help function: `help(ts.google)` to decide on which arguments we require. The more common ones are listed below.

- **query_text**: this is the default string we are translating
- **from_language**: this defaults to 'auto' and uses the apo to 'guess' the language
- **to_language**: by default, all translations are to English (`en`)

- **sleep_seconds:** If doing multiple requests it may be worth spacing these out to prevent flooding the server and getting them rejected.

Translating a Single Phrase

We begin with the common test phrase 'The fox jumped over the lazy dog' and translate this to Welsh `(cy)` :

```
phrase = 'The quick brown fox jumps over the lazy dog.'
ts.google(phrase, from_language='en', to_language='cy')
```

This returns: *"Mae'r llwynog brown cyflym yn neidio dros y ci diog."*

Since I do not speak Welsh, I can test the sentiment of the translation by then translating it back:

```
phrase = "Mae'r llwynog brown cyflym yn neidio dros y ci diog."
ts.google(phrase)
```

We get a translation very similar to the original string: "The fast brown fox jumps over the lazy dog."

Note: *Much like Chinese whispers, there are certain nuances of language that are often lost in translation. This means we are very unlikely to get the exact original phrase.*

character limit.

As an example, we take a couple of lines from Douglas Adam's Hitchhiker's Guide to the Galaxy:

```
corpus = ["On display? I eventually had to go down to the cellar to
find them.",

"That's the display department.",

"With a flashlight.",

"Ah, well the lights had probably gone.",

"So had the stairs.",

"But look, you found the notice didn't you?",

"Yes, said Arthur, yes I did. It was on display in the bottom of a
locked filing cabinet stuck in a disused lavatory with a sign on the
door saying 'Beware of the Leopard'"]
```

We can now translate them in turn using:

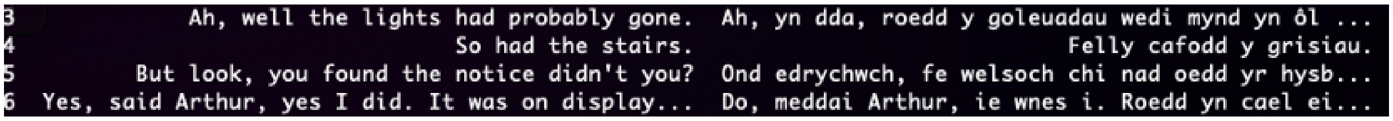
```
welsh = [ts.google(phrase, from_language='en', to_language='cy') for
phrase in corpus]
```

Translating a DataFrame

In the case our information is contained within a pandas data frame, we select an individual column and apply the translate function to it:

```
import pandas as pd
df = pd.DataFrame(corpus, columns = ['text'])

df['welsh_text'] = df['text'].apply(lambda x: ts.google(x,
from_language='en', to_language='cy'))
```



```
3            Ah, well the lights had probably gone.  Ah, yn dda, roedd y goleuadau wedi mynd yn ôl ...
4                                 So had the stairs.                       Felly cafodd y grisiau.
5         But look, you found the notice didn't you?  Ond edrychwch, fe welsoch chi nad oedd yr hysb...
6  Yes, said Arthur, yes I did. It was on display...  Do, meddai Arthur, ie wnes i. Roedd yn cael ei...
```

Example output of above commands.

Page translation: HTML / XML

Often it is not just as simple as having a text document we need to translate. For instance, we may have information on a website that needs to be available to another country or group. For such a situation we can use BeutifulSoup to extract our elements and replace them in situ.

Installation

To do this we install Beautiful Soup 4:

```
pip install bs4
```

Beautiful Soup Documentation - Beautiful Soup 4.4.0 documentation

Beautiful Soup 4 is published through PyPi, so if you can't install it with the system packager, you can install it...

beautiful-soup-4.readthedocs.io

Sample HTML template

```
html = '''

<html>

<head>

<title>That is the display section.</title>

</head>

<body>

<p>So got the stairs.</p>

<p>Yes, said Arthur, yes I did. It was displayed at the bottom of the
locked filing cabinet stuck in a dormant toilet with a sign on the
door saying 'Be wary of the leopard'</p></body>

</html>

'''
```

Scripting Beautiful Soup

We start by deciding on which elements contain our text. For this example it is the paragraph and title fields only, however, these could be marked using attributes or classes instead.

We then read the code (our soup), extract these elements and use the `Navigatable String` property and apply an in-place `replace_with` to substitute our translation.

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'html.parser')

elements = ['p','title']

for i in soup.findAll(elements):
    i.string.replace_with(ts.google(i.string,from_language='cy'
```

Running this returns the original document with the `p` and `title` elements translated:

```
<html>

<head>

<title>That is the display section.</title>

</head>

<body>

<p>So got the stairs.</p>

<p>Yes, said Arthur, yes I did. It was displayed at the bottom of the
locked filing cabinet stuck in a dormant toilet with a sign on the
door saying 'Be wary of the leopard'</p></body>

</html>
```

Conclusion

In this tutorial we covered the use of several language translation APIs through python, and how to translate text within a data frame. Finally, we looked at using Beautiful Soup to extract text from an XML template and produce a translation in place. Hopefully, this tutorial helps make the text more accessible.