PythonによるセマンティックTEIマークアップのためのガイドライン

ver 1.0: 最終更新 2019年1月28日

小風尚樹

目次

1. はじめに

1-1. 何を扱うのか

本ガイドラインは、**プログラミング言語Python**を用いて、TEIマークアップを効率化・半自動化できるようにすることを技術的な目的としています。

歴史研究に資するようなタグ付けを行うことを想定し、青空文庫からアクセスできる服部之総「黒船来航」をマークアップのサンプルテクストに選びました。 その理由は、分量が適度であること・オープンアクセスであること・人名などの固有表現が複数種類見られること、です。なお今回は、XHTMLファイルではなくプレーンテクストに基づいてマークアップを行うこととします。

本ガイドラインは、日本語テクストを対象としたTEIマークアップの実践例の一つとして位置づけられるでしょう。具体的には、2016年6月にTEIコンソーシアム内に設置された東アジア／日本語分科会の成果発信

の一つになります（永崎, 2017, p. 66）。

1-2. 何を扱わないのか

まず、すべてのマークアップ工程をプログラミングで代替するわけではありません。手作業で行う工程を残しつつ、作業の大部分を自動化することを目指します。

また、TEIヘッダーに記述すべき書誌情報や電子校訂版の作成に関する編集情報についても扱いません。

本ガイドラインは、TEIおよびPythonの基礎的な理解を前提とし、それぞれの技術的解説などについては基本的に割愛します。本ガイドラインで扱うPythonプログラムの理解については、例えば (Sweigart, 2017) の第I部「**Pythonプログラミングの基礎**」および第8章「**ファイルの読み書き**」、第11章「**Webスクレイピング**」、第14章「**CSVファイルとJSONデータの操作**」が役に立つでしょう。

最も重要な論点として、TEIヘッダーの**<particDesc>**エレメント内で説明すべき、本文中で言及される人物や地名のマークアップについても扱いません。本ガイドラインでは、次節で説明するように、TEIファイルの外部に存在する学術ソースへのリンク付けを重視し、同一TEIファイル内での相互ID参照については考察の対象外とします。とは言え、同一ファイル内での相互ID参照も重要なマークアップ実践ですので、こちらの解説については (Eide, 2014, chap. 4: Interconnections) に譲りたいと思います。

1-3. より広い意義：セマンティックWEBとTEIマークアップ

本ガイドラインでは、人名・地名などの固有表現を形態素解析モジュールによって抽出します。この固有表現抽出のことを英語ではNER (Named Entity Recognition) と呼びますが、NER技術を活用してマークアップ支援を行うオンライン・プラットフォームとして、Pelagios CommonsによるRecogitoが挙げられます (Simon et al., 2015)。本ガイドラインは、技術レベルではこのようなサービスに劣りますが、TEIマークアップによって生み出される利点についてのビジョンを共有しています。

そのビジョンとは、すなわち、テクストに見られる固有表現について可能な限り外部の学術ソースへのリンクを貼ることによって、自らがマークアップしたテクストをセマンティックWEB技術によって解釈しやすいデータとして流通させることができ、結果としてウェブ上に広がる膨大な知のネットワークを豊かにすることができる、というものです (Bizer, 2009; Hughes et al., 2016, pp. 160-161; Oldman et al., 2016, p. 262)。

このようなビジョンを前提に、本ガイドラインにおける具体的なマークアップ実践としては、地名・人名辞典へのリンク付けを重視します。その際、それぞれ次のようなソースへのリンク付けを検討します。

- 地名：GeoNames
- 人名：VIAF

TEIマークアップは、テクストの読解・解釈にあたって、外部の学術情報にアクセスしながら自分の理解を促進するという側面がある一方で、適切なタグ選定やリンク付けによって、ウェブの世界へ知を発信するという側面も兼ね備えているのです。

このような性質から、TEIマークアップをする上ではウェブ上の情報にアクセスすることが手続き上は重要になりますので、Pythonなどのプログラミング言語のサポートを受けることが有効です。

1-4. 本ガイドラインの構成

まず第I部では、自動化処理のための事前準備について触れます。テクスト中の、**時間・地名・人名・歴史的事象に関する記述**を**形態素解析のためのmecab-python3モジュール**を用いて半自動的に抽出し、**Pythonのリスト型のデータ**として格納しておきます。また、それらの記述をマークアップするにあたって必要となるTEIのタグや属性についても検討します。

次に第II部では、第I部で用意しておいたデータセットに基づいて、**PythonのBeautiful Soup4モジュールを用いたウェブスクレイピングや、WEB APIへのアクセスとJSONデータの処理、Pythonのforループ文を用いた文字列型オブジェクトの置換処理**を行います。

I. 下準備

2. 時間の記述

2-1. 該当記述のリストアップ（手作業）

「黒船来航」のテクストには、時間（ここでは年月）に関する記述として以下のものが見られます。

```
一八四〇–四二；一八四八年
```

そこで、上記の記述を、**Pythonのリスト型データ**として格納しておきます。第II部では、このリスト内のデータを、TEIのタグに囲まれたテクストとして置換する、という処理を行います。なお、この程度なら手作業でマークアップしてもほとんど労力は変わりませんが、本ガイドラインで検討する考え方は、より大規模なテクストデータを扱う際にも有効です。

```
dates = ['一八四〇–四二', '一八四八年']
```

2-2. TEIタグセットの検討

TEIでは、年月を記述するためのタグとして、**<date>**を用意しています（http://www.tei-c.org/release/doc/tei-p5-doc/ja/html/ref-date.html ）。

前節で見たように、サンプルテクストでは年代の範囲と西暦年が記述されていることがわかりますので、例えば **@when, @from, @to** 属性を用いて下記のようにマークアップすることになります。

※なお、これらの属性に入れる値としては、W3C (World Wide Web Consortium) の *XML Scheme Part 2: Datatypes Second Edition* に従うことが推奨されています。標準的な形式でデータを記述することは、コンピュータによる処理を容易にするため、TEIにおける時間の記述においても、W3Cなどの定めるデータフォーマットが望ましいとされています (Wittern et al., 2009, p. 288)。

```
<date from="1840" to="1842">一八四〇–四二</date>
<date when="1848">一八四八年</date>
```

これが仮に、清朝中国の暦や和暦で記述されていた場合は、次のように **@when-custom, @from-custom, @to-custom** 属性を、**@datingMethond, @calendar** 属性と併用すると良いでしょう。

```
<date datingMethod="#japanese" calendar="#japanese" from-custom="天保:10" to-custom="天保:12">天保十年–天保十二年</date>
<date datingMethod="#qing_dynasty" calendar="#qing_dynasty" when-custom="道光:28">道光二十八年</date>
```

上記の **@when-custom** 属性などの値の区切り値として、半角コロンが用いられていますが、TEI P5ガイドライン上では特に指定はなく、プロジェクト内で一貫した記述をすることだけが推奨されています（http://www.tei-c.org/release/doc/tei-p5-doc/ja/html/ref-att.datable.custom.html#tei_att.when-custom ）。

---

3. 人名の記述

3-1. 人名のリストアップ（手作業）

「黒船来航」のテクストには、人名に関する記述としてどのようなものが見られるでしょうか。もちろん、手作業でリストアップしていくことも可能ですが、ここでは自然言語処理技術のひとつ形態素解析のサポートを受けてみたいと思います。

詳しい解説は、(Bird et al., 2010; 柳井・庄司, 2019) などに譲り、ここでは**mecab-python3モジュール**を用いて、ごく簡単に人名の記述を取り出してみたいと思います。モジュールのダウンロードなどについては、開発者のGitHubページをご覧ください（https://github.com/SamuraiT/mecab-python3 ）。各コードの目的などについては、コメント行を参照ください。

次章でも、地名に関する記述について**mecab-python3モジュール**を活用し、半自動的に地名のリストを作成することになります。人名のリストアップを行うコードは次のようになるでしょう。

```
import MeCab

contents = """
中国制覇の足がかり

　阿片あへん戦争（一八四〇–四二）で中国が開国した後は極東の一角日本を開けばこれで旧文明国を資本主義世界に開放する事業が完成するわけである。・・・（以下略）・・・"""

m = MeCab.Tagger()
parsed_words = m.parse(contents)
parsed_list = parsed_words.split('\n')

# parsed_list[130] = '林\t名詞,固有名詞,人名,姓,*,*,林,ハヤシ,ハヤシ'
# 人名は、「名詞,固有名詞,人名」と形態素解析されることがわかるので、
# 形態素が「人名」の値を持っていたら、人名のリストに格納するようにする

people = []
```

```
for row in parsed_list:
    word = row.split('\t')[0]
    if word == 'EOS':
        break
    else:
        pos = row.split('\t')[1].split(',')[2]
        if pos == '人名':
            people.append(word)
```

重複があるので、ユニークな値のみ残したリストを作成します。

```
people = list(set(people))

# people = ['えい', '広中', '昌益', '長英', '象山', '河野', 'つぎ', '井伊', '佐久間',
'兵衛', '生麦', '陸奥', '嶺山', '浜口', '渡辺', '眠', '金', '佐藤', 'こう', '安藤',
'のぶ', '信淵', '子平', 'ペリー', '巧', '高野', '仲', '橋本', '林', '宗光']
```

人名のリストについては、姓名を結合した形でリストに登録するなど改善の余地が考えられます。上記のコードを参考にしながら、発展課題として考えてみてください。修正した人名リストを以下に示します。

```
persons = ['林子平', '橋本左内', 'ペリー', '井伊大老', '安藤昌益', '佐藤信淵', '渡辺崋
山', '高野長英', '佐久間象山', '浜口梧陵', '林金兵衛', '河野広中', 'グラント将軍', '陸
奥宗光']
```

3-2. TEIタグセットの検討

人名のマークアップ例は、次のようになるでしょう。

```
<persName ref="http://viaf.org/viaf/42142122">林子平</persName>
```

TEIマークアップ中で何かを参照しようとする際に用いられる属性は、上記の **@ref** 属性のほかにも、**@key** 属性が挙げられます。TEI P5ガイドラインの att.canonical 属性クラスの解説によれば、**@ref** 属性での参照はより明示的で、外部で定義されたURIなどを指定する際に用いる一方、**@key** 属性は何らかの決まりにしたがった形で外部参照の手段を提供する際に用いるとされています (http://www.tei-c.org/release/doc/tei-p5-doc/ja/html/ref-att.canonical.html)。

つまり、**@key** 属性の方が、プロジェクト毎のルールに沿った柔軟な参照方法を記述できるわけですが、本ガイドラインでは**@ref** 属性を用いて外部のURIを直接記述する方が適切であるということになります。

---

4. 地名の記述

4-1. 地名のリストアップ（半自動）

「黒船来航」のテクストには、地名に関する記述としてどのようなものが見られるでしょうか。こちらについても、形態素解析のサポートを借りることから始めましょう。

```
import MeCab

# 「黒船来航」のテクスト全文を変数に格納しておく
contents = """
中国制覇の足がかり

　阿片あへん戦争（一八四〇–四二）で中国が開国した後は極東の一角日本を開けばこれで旧文明国を資本主義世界に開放する事業が完成するわけである。・・・（以下略）・・・"""

m = MeCab.Tagger()
parsed_words = m.parse(contents)
parsed_list = parsed_words.split('\n')

# parsed_list[0] = '中国\t名詞,固有名詞,地域,国,*,*,中国,チュウゴク,チューゴク'
# 地名は、「名詞,固有名詞,地域」と形態素解析されることがわかるので、
# 形態素が「地域」の値を持っていたら、地名のリストに格納するようにする

places = []

for row in parsed_list:
    word = row.split('\t')[0]
    if word == 'EOS':
        break
    else:
        pos = row.split('\t')[1].split(',')[2]
        if pos == '地域':
            places.append(word)

# 重複を削除
places = list(set(places))

# places = ['台湾', '上海', '小笠原', 'アメリカ', 'ナンキン', '極東', '英', 'さつま', '中国', '文久', 'アジア', 'ポーツマス', 'イギリス', 'フランス', 'インド', '薩摩', '琉球', '露', '那覇', '日', '米', '尾張', '朝鮮', '日本', '南京', 'シャンハイ', 'サンフランシスコ', '西欧', '父島', '対馬']
```

ただし、ここにはリストアップされていない「紀州」などの地名もありますし、このリストには読み仮名が登録されていたりするなど、改善の余地が残されています。mecab-python3の辞書にユーザー定義の単語を登録することもできますが、今回は手作業で地名リストを調整することにしておきます。検討を要する地名もありますが、ひとまず次に進みましょう。

```
places = ['中国', '日本', '極東', 'イギリス', 'インド', 'アメリカ', 'サンフランシスコ', '沖縄', '那覇', '小笠原', '父島', '上海', '生麦', '薩摩', '対馬', '紀州', '尾張', 'フランス', 'アジア', '台湾', '朝鮮', 'ポーツマス']
```

4-2. TEIタグセットの検討

前節で検討した記述には、「サンフランシスコ」などのように都市や行政区画を表す語が見られる一方、「アジア」などのように明確な行政的領域というよりは、概念上の地政学的領域を示した語も見られます。TEIガイドラインでは、このような様々な種類の「地名」をマークアップするにあたって、**<placeName>**タグのほかにも、様々なタグセットを用意しています（http://www.tei-c.org/release/doc/tei-p5-doc/ja/html/ND.html#NDPLAC ）。

しかしながら、GeoNames などの地名辞典の情報と組み合わせて地理的範囲を示すことを考えると、国名や地域名などのように時代や見解によって解釈の余地が分かれる地名を詳しくマークアップすることは、時として論争的かつ政治的な側面を持つことでしょう。少なくとも本ガイドラインの第II部では、GeoNames にデータが掲載されていれば、下記のように外部リンクを付与しておきたいと思いますが、実際のマークアップでは、このような機械的なタグ付けの結果に対して、参照先のデータが妥当であるかどうか検討する必要があることを念頭に置いてください。

```
<placeName ref="https://www.geonames.org/5391959">サンフランシスコ</placeName>
```

@ref 属性を用いている理由については、3-2. をご参照ください。

4-3. 地名辞典について

なお、歴史的地名や地理的領域に関しては、様々な地名辞典が存在します。日本の地名に関して言えば、平凡社『日本歴史地名体系』や、人間文化研究機構およびH-GIS研究会により公開された「歴史地名データ」などを挙げることができます。海外地名に関しては、「1. はじめに」で紹介したRecogitoが、1492年以前の歴史地名に関する地名辞典を複数参照してアノテーションを行う機能を提供しています。

---

5. 歴史的事象の記述

5-1. イベントのリストアップ（手作業）

「黒船来航」のテクストには、歴史的事象に関する記述として少なくとも以下のものが見られます。

```
阿片戦争；南京条約；生麦事件；南北戦争；明治維新；台湾征伐；日露戦争；サンフランシスコ条約；日米安全保障条約；行政協定；日米通商航海条約
```

それでは、これらの記述も**Pythonのリスト型データ**として格納しておきましょう。

```
events = ['阿片戦争', '南京条約', '生麦事件', '南北戦争', '明治維新', '台湾征伐', '日露戦争', 'サンフランシスコ条約', '日米安全保障条約', '行政協定', '日米通商航海条約']
```

5-2. TEIタグセットの検討

意外なことに、歴史的事象、より広義にはイベントについては、人名や地名と違い、本文のマークアップで使用することのできる **<eventName>** というタグが用意されていません。代わりに、**<name>**タグの中の **@type** 属性の値として event と記述することで代替すれば良いと考えられているためです (Ore & Eide, 2009, pp. 167-169)。より明示的なデータ記述ができるようにTEIガイドラインが改訂されてきていることを考えれば、このようなタグが導入される可能性も考えられます。

<name> タグとは別に、**<rs>** (referring strings) タグを用いることも考えられます。これらのタグは、TEI P4ガイドラインにおいて、固有表現のマークアップのために用いられていました (Eide, 2014, chap. 2: P4 to P5)。本ガイドラインでは、**<name>** を用いておきたいと思います。

```
<name type="event">阿片戦争</name>
```

II. 自動化処理

本セクションでは、第I部で手作業で**Pythonのリスト型データ**として格納しておいた記述を対象に、検討したTEIタグセットを用いてマークアップするという作業を自動化します。基本的には、**Pythonを用いた検索・置換**を行います。

まず最初に、すべての自動化処理に共通する準備として、**Pythonプログラムから「黒船来航」のプレーンテクストファイルを読み込んで、文字列型のオブジェクトとして変数に格納**しておきたいと思います。

```
input_file = open('aozora_kurofune.txt', 'r', encoding='utf-8')
whole_text = input_file.read()
input_file.close()
```

6. 時間のマークアップ

6-1. 漢数字をアラビア数字に変換する

この変換には様々な方法があると思いますので、ご自身でサンプルコードを改善してみてください。本ガイドラインでの手順は、次の通りです。

- 漢数字とアラビア数字の対応辞書を作成する
- 年月日のリストを対象に、漢数字をアラビア数字に置換する
- 漢数字とアラビア数字が対になった辞書を定義する

```
convert_num_dict = {'〇':'0', '一':'1', '二':'2', '三':'3', '四':'4', '五':'5',
'六':'6', '七':'7', '八':'8', '九':'9'}

dates = ['一八四〇–四二', '一八四八年']

num_pair_dict = {}
```

```
for date in dates:
        arabic = date
        for character in date:
            if character in convert_num_dict.keys():
                arabic = arabic.replace(character, convert_num_dict[character])
        num_pair_dict[date] = arabic

# num_pair_dict = {'一八四〇–四二': '1840–42', '一八四八年': '1848年'}
```

6-2. 本文の該当箇所をタグ付きで置換する

```
for kansuji, arabic in num_pair_dict.items():
        if '–' in arabic:
            dateRange = arabic.split('–')
            start = dateRange[0]
            end = dateRange[1]
            encoded_date = '<date from="{start}" to="{end}">{kansuji}</date>'
            whole_text = whole_text.replace(kansuji,
encoded_date.format(start=start, end=end, kansuji=kansuji))
        elif '年' in kansuji:
            arabic = kansuji.replace('年','')
            encoded_date = '<date when="{arabic}">{kansuji}</date>'
            whole_text = whole_text.replace(kansuji,
encoded_date.format(arabic=arabic, kansuji=kansuji))
```

上記のPythonプログラムにも見られるように、置換処理によるテクストマークアップには、**Pythonのformat記法**が非常に有効です (https://docs.python.jp/3/library/string.html#custom-string-formatting)。

---

7. 人名のマークアップ

7-1. VIAFに記載されたURIを機械的に取得する

VIAFに記載された人名のIDを取得するには、どのようにすれば良いでしょうか。例えば、「林子平」のVIAF IDを手作業で取得したい場合、

1. VIAFのトップページにアクセスし、検索窓に「林子平」と入力する
2. 検索結果が表示されるので、適切なものを判断してクリックする
3. クリックした先で表示されているVIAF IDをコピー＆ペーストする

というような手順を踏むはずです。この一連の作業をPythonに手伝ってもらうとすると、例えば次のような処理をすれば良さそうです。

1. **PythonのRequestsモジュールで、「林子平」とVIAF上で検索した結果出力されるHTMLファイルを取得する**
2. **PythonのBeautiful Soup4モジュールで、検索結果を指定する**

3. **PythonのBeautiful Soup4モジュールで、検索結果のVIAF IDをHTMLソースから取得する**

ただし、HTMLファイルを対象としたWEBスクレイピングは、ウェブページのデザインが変更されたりするとスクレイピングのコードを活用することができなくなってしまう危険性があります。つまり、安定したデータ取得ができないのです。

現在、多くのウェブサービスでは、人間のユーザがブラウザで閲覧するために生成されたHTMLではなく、コンピュータによる分析や二次利用に適したデータだけが格納されたXMLやJSONファイルを、WEB APIとして提供しています。WEB APIを公開することにより、アプリケーション同士の情報の共有・分析・再利用のスピードが格段に向上するため、結果としてAPIを公開しているウェブサービスの価値を高めることにつながるのです (水野, 2014; Blanke, 2014)。この背景には、標準化された通信プロトコルであるHTTPを介して、ウェブリソースのアドレスをURIによって一意に指定できるようにWEB APIを設計することにより、ウェブ上のデータが相互につながりあい、結果としてウェブの世界が成長するというREST APIの設計理念を見て取ることができます (Fielding, 2005, chap. 5: Representational State Transfer (REST))。

WEB APIは、ウェブ空間の性格を大きく変えることに貢献した技術のひとつです。すなわち、かつてのWWW（ワールド・ワイド・ウェブ）においては、サービスを提供する側とサービスを受け取るユーザの関係が非対称的で一方向的なものだったのですが、WEB APIをはじめとするWEB 2.0技術の登場により、WWW上ではユーザがウェブ上に流通するデータに容易にアクセスし、自らも価値を発信できるようになりました。すなわち、サービス提供のためのプラットフォームから情報発信のためのプラットフォームへと変貌を遂げたのです (O'Reilly, 2005; 橋本, 2018)。TEIマークアップを通じてウェブの世界に知を発信することも、XMLというコンピュータで扱いやすい標準化されたデータ形式に則っていることに大きく支えられています (Ide et al., 2019)。

本ガイドラインで扱うVIAFも、WEB APIを提供していますので、次のような処理手順を踏むことにしましょう。

1. **Pythonのrequestsモジュールで、VIAFのAPIに対して人名に基づいたクエリを投げる**
2. **返されたJSONデータを、PythonのJSONモジュールでPythonの辞書型オブジェクトに変換する**
3. **VIAF IDを指定して取得する**

---

1. 人名に基づくクエリを投げる

VIAFのAPIドキュメントによれば、次のようにURIを指定することによって、人名による検索結果を複数件格納したJSONデータとして取得できます (https://platform.worldcat.org/api-explorer/apis/VIAF)。

```
http://www.viaf.org/viaf/AutoSuggest?query=austen
```

このURIのうち、「austen」と記述されている箇所だけを変更すれば、検索したい人物を変更することができます。そこで、事前に準備しておいた人名リストに基づいて、上記のURLの名前の部分だけが変わるように**forループ文**を書きます。

```
import requests

persons = ['林子平', '橋本左内', 'ペリー', '井伊大老', '安藤昌益', '佐藤信淵', '渡辺崋
山', '高野長英', '佐久間象山', '浜口梧陵', '林金兵衛', '河野広中', 'グラント将軍', '陸
```

```
奥宗光']

viaf_query_uri = 'http://www.viaf.org/viaf/AutoSuggest?query={person_name}'

for person in persons:
    query_result = requests.get(viaf_query_uri.format(person_name=person))
```

2. JSONデータを辞書型オブジェクトに変換

返されたJSONデータを確認する前に、ひとまずJSONデータを辞書型オブジェクトに変換します。

```
import json

api_json = query_result.text
api_dict = json.loads(api_json)
```

3. VIAF IDの取得

JSONデータの構造を確認するには、例えばFireFoxブラウザが色分けと整形を行ってくれますので、活用してみると良いでしょう。


```
query:          "林子平"
result:
  0:
    term:       "林子平"
    displayForm: "林子平"
    nametype:   "personal"
    lc:         "n82000980"
    dnb:        "124834817"
    bnf:        "15138219"
    viafid:     "42142122"
    score:      "2456"
    recordID:   "42142122"
  1:
    term:       "林子平, 1738-1793"
    displayForm: "林子平, 1738-1793"
    nametype:   "personal"
    lc:         "n82000980"
    dnb:        "124834817"
    bnf:        "15138219"
    viafid:     "42142122"
    score:      "1202"
    recordID:   "42142122"
  2:
    term:       "林子平, 1738-1793. | 海國兵談 | German |"
    displayForm: "林子平, 1738-1793. | 海國兵談 | German |"
    nametype:   "uniformtitleexpression"
    lc:         "n2004038285"
    viafid:     "177957769"
    score:      "1016"
    recordID:   "177957769"
  3:
    term:       "林子平, 1738-1793. | 海國兵談"
    displayForm: "林子平, 1738-1793. | 海國兵談"
    nametype:   "uniformtitlework"
    viafid:     "750715147504170049004"
    score:      "1011"
    recordID:   "750715147504170049004"
  4:
    term:       "林子平, 1923-"
    displayForm: "林子平, 1923-"
    nametype:   "personal"
    lc:         "n93103628"
    nla:        "000036668545"
    viafid:     "23817100"
    score:      "644"
    recordID:   "23817100"
```

複数階層の辞書やリストで構成されたJSONデータを掘り下げていくと、人名とVIAF IDが格納された辞書を見つけることができます。例えば1番目の検索結果の人名とVIAF IDを取得するには、次のように指定しましょう。

```
person = api_dict['result'][0]['term']
viafID = api_dict['result'][0]['viafid']

# viafID = '' となる
```

さて、この後、最低限必要な箇所だけ抽出して、最終的に絶対パスに変換します。人名とVIAF IDの組み合わせを辞書型オブジェクトとして格納しておきます。

```
viaf_uri = 'https://viaf.org/viaf/' + viafID
# viaf_uri = 'https://viaf.org/viaf/42142122' となる

persons = {'林子平': 'https://viaf.org/viaf/42142122'}
```

今回のように、検索語が一意に特定できるような人名である場合には、1番目の検索結果を取得しても問題ないかもしれませんが、「ペリー」などのように様々な人物を示す可能性のある人名のVIAF IDを検討するために、手作業を経由すると良いでしょう。この点については、本ガイドラインの第III部にまとめたモジュールの中に記述してありますので、ご確認ください。

7-2. 本文の該当箇所をタグ付きで置換する

人名リストに対応するVIAF IDを取得した後は、**本文テクストをタグ付きマークアップテクストとして置換する処理**を書きます。該当箇所の記述は次のようになるでしょう。

```
for person, uri in persons.items():
    encoded_personName = '<persName ref="{uri}">{person}</persName>'
    whole_text = whole_text.replace(person, encoded_personName.format(uri=uri,
person=person))
```

8. 地名のマークアップ

8-1. GeoNamesに記載されたURIを機械的に取得する

GeoNamesに記載された地名のIDを取得するには、どのようにすれば良いでしょうか。もちろん、前章のようにGeoNamesのWEB APIにアクセスすることが望ましいのですが、GeoNames APIの利用にはユーザ認証が必要ですので (http://www.geonames.org/export/web-services.html)、各自試してみてください。

そこで今回は、あまり汎用的なコードにはなりませんが、HTMLファイルからのスクレイピングについて扱ってみたいと思います。例えば、「サンフランシスコ」のGeoNames IDを手作業で取得したい場合、

1. GeoNamesのトップページにアクセスし、検索窓に「サンフランシスコ」と入力する
2. 検索結果が表示されるので、適切なものを判断してクリックする
3. クリックした先で表示されているGeoNames IDをコピー＆ペーストする

というような手順を踏むはずです。この一連の作業をPythonに手伝ってもらうとすると、例えば次のような処理をすれば良さそうです。

1. **PythonのRequestsモジュールで、「サンフランシスコ」とGeoNames上で検索した結果出力されるHTMLファイルを取得する**
2. **PythonのBeautiful Soup4モジュールで、今回は便宜的に一番上の検索結果を選択する**
3. **PythonのBeautiful Soup4モジュールで、検索結果のGeoNames IDをHTMLソースから取得する**

では、この手順にしたがって、Pythonプログラムの構成を検討していきます。

1. 検索結果のHTMLファイル取得

GeoNamesのトップページで、「サンフランシスコ」と検索すると、次のようなURLへ遷移します。

https://www.geonames.org/search.html?q=サンフランシスコ&country=

このURLのうち、「サンフランシスコ」と記述されている箇所だけを変更すれば、検索したい地名を変更することができると考えられます。そこで、事前に準備しておいた地名リストに基づいて、上記のURLの名前の部分だけが変わるように**forループ文**を書けば良いでしょう。

```
import requests

places = ['中国', '日本', '極東', 'イギリス', 'インド', 'アメリカ', 'サンフランシスコ', '沖縄', '那覇', '小笠原', '父島', '上海', '生麦', '薩摩', '対馬', '紀州', '尾張', 'フランス', 'アジア', '台湾', '朝鮮', 'ポーツマス']

geoNames_query_url = 'https://www.geonames.org/search.html?q={place_name}&country='

for place in places:
    query_result = requests.get(geoNames_query_url.format(place_name=place))
```

2. 1番目の検索結果を選択する

上記URLのHTMLソースコードを確認してみると、検索結果が <table class="restable"> タグの内容として提供されていることがわかります。

```
<table class="restable">
    <tr><td colspan=6 style="text-align: right;"><small>3 records found for "サンフランシスコ"</small></td></tr>
    <tr><th></th><th>Name</th><th>Country</th><th>Feature class</th><th>Latitude</th><th>Longitude</th></tr>
    <tr><td><small>1</small> <a href="/5391959/san-francisco.html"><img src="/maps/markers/m10-WHITE-P.png" border="0" alt="P"></a></td>
```

ただし、上記からわかるように、GeoNamesのIDが記述されている <a> タグの @href 属性付近には、HTMLタグにIDやクラスが充分に付されていないことに注意する必要があります。いくつか指定方法はありますが、1番目の検索結果の <a> タグを選択するために、**<table class="restable"> タグに注目し、その子要素の3番目の <tr> タグの子要素として、目当ての <a> タグを選択します**。該当箇所のコードは次のようになるでしょう。

```
soup_object = bs4.BeautifulSoup(query_result.text, 'lxml')
table_list = soup_object.select('.restable')
tr_tags = table_list[0].select('tr')
target_a_tag = tr_tags[2].a
```

3. GeoNames IDの取得

取得したいGeoNames IDは、<a> タグの @href 属性の値として相対パスで記述されていますので、**この値を取得した後、最低限必要な箇所だけ抽出して、最終的に絶対パスに変換**します。

```
geoNames_id_relative_path = target_a_tag.attrs['href']
# geoNames_id_relative_path の値は、'/5391959/san-francisco.html'

id_path_list = geoNames_id_relative_path.split('/')
uri_endpoint = '/'.join(id_path_list[:2])

geoNames_id = 'https://www.geonames.org' + uri_endpoint
# geoNames_id の値は、'https://www.geonames.org/5391959' となる
```

以上で、placesという変数名の地名リストに格納されたテクストデータひとつひとつに対して、対応するGeoNames ID（仮）を取得することができました。

このGeoNames IDが正しいのかどうかについては、今回は手作業で確認するタスクとして残しておきたいと思います。

8-2. 本文の該当箇所をタグ付きで置換する

地名リストに対応するGeoNames上のIDを取得した後は、**本文テクストをタグ付きマークアップテクストとして置換する処理**を書きます。該当箇所の記述は次のようになるでしょう。

```
for place in places:
    encoded_placeName = '<placeName ref="{uri}">{place}</placeName>'
    whole_text = whole_text.replace(place,
encoded_placeName.format(uri=geoNames_id, place=place))
```

---

9. 歴史的事象のマークアップ

今回は、特に歴史的事象に関する外部ソースを参照していないので、単純に**テクストの置換**になります。

```
events = ['阿片戦争', '南京条約', '生麦事件', '南北戦争', '明治維新', '台湾征伐', '日露戦争', 'サンフランシスコ条約', '日米安全保障条約', '行政協定', '日米通商航海条約']

for event in events:
    encoded_event = '<name type="event">{event}</name>'
    whole_text = whole_text.replace(event, encoded_event.format(event=event))
```

III. サンプルPythonモジュール

これまで解説してきたPythonプログラムを、ひとつのモジュールとしてまとめたものを掲載しておきます。言及していない論点について、一覧にしておきます。

- **GeoNamesやVIAFを対象にウェブスクレイピングを行うので、timeモジュールのsleep関数を使って、相手サーバーの負担を減らす工夫をした**
- **GeoNamesやVIAFに該当記述がない場合の例外処理を組み込んだ**
- **APIでの検索結果を複数件取得し、IDの妥当性を検証する手作業をサポートする関数を組み込んだ**
- **最終的に置換した文字列を、<body> タグで囲んだ**

```
import MeCab, requests, bs4, time, json

dates = ['一八四〇–四二', '一八四八年']

events = ['阿片戦争', '南京条約', '生麦事件', '南北戦争', '明治維新', '台湾征伐', '日露戦争', 'サンフランシスコ条約', '日米安全保障条約', '行政協定', '日米通商航海条約']


def extract_people_and_places(target_text):
    """
    形態素解析によって人名と地名を抽出し、それぞれリストとして返す
    """

    m = MeCab.Tagger()
    parsed_words = m.parse(target_text)
    parsed_list = parsed_words.split('\n')

    people = []
    places = []

    for row in parsed_list:
        word = row.split('\t')[0]
        if word == 'EOS':
            break
        else:
            pos = row.split('\t')[1].split(',')[2]
            if pos == '人名':
```

```
                people.append(word)
            elif pos == '地域':
                places.append(word)

    # 重複を削除
    people = list(set(people))
    places = list(set(places))

    print('Finished parsing and listing up the people and places.')

    # 人名・地名リストをタプルに格納して返す
    return people, places


def make_file_to_normalize_entities(filename, entity_list):
    """
    固有表現の正規化を手作業で行うためのファイルを返す
    """

    output_file = open(filename, 'w', encoding='utf-8')
    for entity in entity_list:
        output_file.write(entity)
        output_file.write('\n')

    output_file.close()

    print('Finished creating the file for normalizing.')

    return None


def get_viaf_api(normalized_person_file):
    """
    人名の表記が正規化されたファイルを読み込み、各人名に対応するVIAF APIを取得し、タブ区切りの文字列型オブジェクトとして返す
    """

    input_file = open(normalized_person_file, 'r', encoding='utf-8')
    normalized_persons = input_file.read().split('\n')

    viaf_query_uri = 'http://www.viaf.org/viaf/AutoSuggest?query={person_name}'

    outputs = ''

    for person in normalized_persons:
        outputs += '========{query}========\n'.format(query=person)

        query_result = requests.get(viaf_query_uri.format(person_name=person))
        api_json = query_result.text
        api_dict = json.loads(api_json)

        if api_dict['result'] != None:
            print('{person} has some search results.'.format(person=person))
            for record_num in range(len(api_dict['result'])):
```

```
                person = api_dict['result'][record_num]['term']
                viafID = api_dict['result'][record_num]['viafid']
                viaf_uri = 'https://viaf.org/viaf/' + viafID
                outputs += '{person}\t{uri}\n'.format(person=person, uri=viaf_uri)

            else:
                print('{person} has no search result.'.format(person=person))
                person = person
                viaf_uri = ''
                outputs += '{person}\t{uri}\n'.format(person=person, uri=viaf_uri)

            time.sleep(0.5)

    print('Completed extracting the search results from VIAF.')

    return outputs


def make_tsv_to_check_api(filename, strings):
    """
    get_viaf_api関数の結果などを書き込んだTSVファイルを返す。手作業での検証作業をサポートする
    """

    output_tsv = open(filename, 'w', encoding='utf-8')
    output_tsv.write(strings)
    output_tsv.close()

    print('Finished writing the strings to the file.')

    return None


def read_tsv_to_make_person_dictionary(filename):
    """
    手作業でVIAF IDを検証したTSVファイルを読み込み、人名とVIAF URIが対になった辞書型オブジェクトを返す
    """

    input_file = open(filename, 'r', encoding='utf-8')
    rows = input_file.read().split('\n')
    person_dictionary = {}

    for row in rows:
        person, viaf_uri = row.split('\t')
        if person not in person_dictionary.keys():
            person_dictionary[person] = viaf_uri

    return person_dictionary


def convert_to_arabic_numeric(target_text, date_list):
    """年月日のリストを対象に、漢数字をアラビア数字に置換した後、漢数字とアラビア数字が対になった辞書を定義し、<date>タグに置換したテクストを返す
```

```
    """

    # (1) まずはアラビア数字に置換し、漢数字とアラビア数字が対になった辞書を作成する

    convert_num_dict = {'〇':'0', '一':'1', '二':'2', '三':'3', '四':'4', '五':'5',
'六':'6', '七':'7', '八':'8', '九':'9'}
    num_pair_dict = {}

    for date in date_list:
        arabic_num = date
        for char in date:
            if char in convert_num_dict.keys():
                arabic_num = arabic_num.replace(char, convert_num_dict[char])
        num_pair_dict[date] = arabic_num

    # num_pair_dict = {'一八四〇–四二': '1840–42', '一八四八年': '1848年'}

    # (2) 上記で作成した辞書を使って、タグ付けを行う
    # 42という記述を1842に修正するのは手作業

    for kansuji, arabic in num_pair_dict.items():
        if '–' in arabic:
            dateRange = arabic.split('–')
            start = dateRange[0]
            end = dateRange[1]
            encoded_date = '<date from="{start}" to="{end}">{kansuji}</date>'
            target_text = target_text.replace(kansuji,
encoded_date.format(start=start, end=end, kansuji=kansuji))
        elif '年' in kansuji:
            arabic = kansuji.replace('年','')
            encoded_date = '<date when="{arabic}">{kansuji}</date>'
            target_text = target_text.replace(kansuji,
encoded_date.format(arabic=arabic, kansuji=kansuji))

    print('Converted Kansuji to Arabic numbers.')

    return target_text


def substitute_to_person_name(target_text, person_dict):
    """人名リストのデータを対象に、テクスト中の該当箇所を<persName>タグで囲んだテクスト
に置換したものとして返す
    """

    for person, uri in person_dict.items():
        encoded_personName = '<persName ref="{uri}">{person}</persName>'
        target_text = target_text.replace(person,
encoded_personName.format(uri=uri, person=person))

        time.sleep(0.5)

    print('Substituted the person names with the encoded texts.')

    return target_text
```

```
def substitute_to_place_name(target_text, place_list):
    """地名リストのデータを対象に、それぞれの記述に該当するGeoNames IDを取得し、
       テクスト中の該当箇所を<placeName>タグで囲んだテクストに置換したものとして返す
    """

    geoNames_query_url = 'https://www.geonames.org/search.html?q=
{place_name}&country='
    for place in place_list:
        query_result = requests.get(geoNames_query_url.format(place_name=place))
        soup_object = bs4.BeautifulSoup(query_result.text, 'lxml')

        if len(soup_object.select('.restable')) >= 1:
            table_list = soup_object.select('.restable')
            tr_tags = table_list[0].select('tr')
            target_a_tag = tr_tags[2].a
            geoNames_id_relative_path = target_a_tag.attrs['href']

            id_path_list = geoNames_id_relative_path.split('/')
            uri_endpoint = '/'.join(id_path_list[:2])

            geoNames_id = 'https://www.geonames.org' + uri_endpoint

        else:
            geoNames_id = ''

        encoded_placeName = '<placeName ref="{uri}">{place}</placeName>'
        target_text = target_text.replace(place,
encoded_placeName.format(uri=geoNames_id, place=place))

        time.sleep(0.5)

    print('Substituted the place names with the encoded texts.')

    return target_text


def add_name_elements_to_events(target_text, event_list):
    """歴史的事象のリストのデータを、<name>タグで囲んだテクストに置換して返す
    """

    for event in event_list:
        encoded_event = '<name type="event">{event}</name>'
        target_text = target_text.replace(event,
encoded_event.format(event=event))

    print('Added name elements with the texts related to events.')

    return target_text



# プレーンテクストの読み込み
```

```
input_file = open('aozora_kurofune.txt', 'r', encoding='utf-8')
whole_text = input_file.read()
input_file.close()

persons, places = extract_people_and_places(whole_text)

# 自動抽出された固有表現を手作業で修正するために、TSVファイルに出力
make_file_to_normalize_entities('NERed_places.tsv', places)
make_file_to_normalize_entities('NERed_persons.tsv', persons)

# 今回、地名リストは'NERed_places.tsv'を基に手作業で修正
normalized_places = ['中国', '日本', '極東', 'イギリス', 'インド', 'アメリカ', 'サンフランシスコ', '沖縄', '那覇', '小笠原', '父島', '上海', '生麦', '薩摩', '対馬', '紀州', '尾張', 'フランス', 'アジア', '台湾', '朝鮮', 'ポーツマス']

# 人名に関しては、'NERed_persons.tsv'を基に手作業で人名表記を正規化したTSVファイル'normalized_people_name.tsv'を作成しておく

# 正規化人名リストを読み込んで、VIAF IDの候補を記したTSVファイルを作成する
viaf_ids = get_viaf_api('normalized_people_name.tsv')
make_tsv_to_check_api('viaf_check.tsv', viaf_ids)

# 'viaf_check.tsv'に基づいてURIの検証を手作業で行い、'verified_viaf.tsv'ファイルとして保存しておく

# 人名とVIAF URIが対になった辞書を作成する
persons_dict = read_tsv_to_make_person_dictionary('verified_viaf.tsv')

"""
persons_dict = {
    '林子平': 'https://viaf.org/viaf/42142122',
    '橋本左内': 'https://viaf.org/viaf/72201183',
    'ペリー': 'http://viaf.org/viaf/37133702',
    '井伊大老': 'http://viaf.org/viaf/42640401',
    '安藤昌益': 'https://viaf.org/viaf/44415205',
    '佐藤信淵': 'https://viaf.org/viaf/60440169',
    '渡辺崋山': 'https://viaf.org/viaf/96574339',
    '高野長英': 'https://viaf.org/viaf/10650351',
    '佐久間象山': 'https://viaf.org/viaf/25948303',
    '浜口梧陵': 'http://viaf.org/viaf/26000155',
    '林金兵衛': 'https://viaf.org/viaf/61383794',
    '河野広中': 'https://viaf.org/viaf/26014043',
    'グラント将軍': 'http://viaf.org/viaf/66505625',
    '陸奥宗光': 'https://viaf.org/viaf/12518180'}
"""

# 定義した関数の実行
whole_text = convert_to_arabic_numeric(whole_text, dates)
whole_text = substitute_to_place_name(whole_text, normalized_places)
whole_text = substitute_to_person_name(whole_text, persons_dict)
whole_text = add_name_elements_to_events(whole_text, events)

# 置換結果の書き込み
result_file = open('result.xml', 'w', encoding='utf-8')
```

```
result_file.write('<body>')
result_file.write(whole_text)
result_file.write('</body>')
result_file.close()
```

IV. サンプルTEIマークアップファイル

本ガイドラインでマークアップしたXMLファイルをもとに、外部ソースへのリンク付けの妥当性などを検証した完成版のTEIマークアップファイルは、TEIコンソーシアムの東アジア／日本語分科会が公開しているGitHubページに掲載してありますので、ご参照ください。

https://github.com/TEI-EAJ/aozora_tei/blob/master/data/complete/tei_lib_lv4/50362_tei.xml

V. 参考文献（アルファベット順）

Bird, S. et al. (2010) 入門自然言語処理. 東京: オライリージャパン.

Bizer, C. et al. (2009) Linked Data: The Story So Far. *International Journal on Semantic Web and Information Systems*. 5 (3), 1–22. Available from: https://eprints.soton.ac.uk/271285/ (Accessed 5 January 2019).

Blanke, T. (2014) *Digital Asset Ecosystems: Rethinking Crowds and Clouds*. Kidlington: Chandos.

Eide, Ø. (2014) Ontologies, Data Modeling, and TEI. *Journal of the Text Encoding Initiative*. [Online] (Issue 8). Available from: http://journals.openedition.org/jtei/1191 (Accessed 30 October 2018).

Fielding, R. (2000) *Architectural Styles and the Design of Network-based Software Architectures*. DPhill Dissertation, University of California, Irvine. [online]. Available from: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (Accessed 27 January 2019).

橋本雄太 (2018) 市民参加型史料研究のためのデジタル人文学基盤の構築. 博士論文, 京都大学. Available from: http://hdl.handle.net/2433/233817 (Accessed 7 December 2018).

Hughes, L. et al. (2016) 'Digital Methods in the Humanities: Understanding and Describing their Use across the Disciplines', in Susan Schreibman et al. (eds.) *A New Companion to Digital Humanities*. 2nd Edition edition Chichester, West Sussex, UK: Wiley-Blackwell. pp. 150–170.

Ide N. et al. (2018) TEI：それはどこからきたのか。 そして、なぜ、今もなおここにあるのか？ デジタル・ヒューマニティーズ. [Online] 1, 3–28. Available from: https://www.jstage.jst.go.jp/article/jadh/1/0/1_2/_article/-char/ja (Accessed 27 January 2019).

水野貴明 (2014) Web API: The Good Parts. 東京: オライリージャパン.

永崎研宣 (2017) デジタル文化資料の国際化に向けて：IIIFとTEI. 情報の科学と技術. 67 (2), 61–66. Available from: http://ci.nii.ac.jp/naid/130005304139 (Accessed 11 January 2019).

Oldman, D. et al. (2016) 'Zen and the Art of Linked Data: New Strategies for a Semantic Web of Humanist Knowledge', in Susan Schreibman et al. (eds.) *A New Companion to Digital Humanities*. 2nd Edition edition Chichester, West Sussex, UK: Wiley-Blackwell. pp. 251–273.

Ore, C.-E. & Eide, Ø. (2009) TEI and Cultural Heritage Ontologies: Exchange of Information? *Literary and Linguistic Computing*. [Online] 24 (2), 161–172. Available from: https://academic.oup.com/dsh/article/24/2/161/978118 (Accessed 15 November 2018).

O'Reilly, T. (2005) *What Is Web 2.0* [online]. Available from: https://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html (Accessed 19 December 2018).

Simon, R. et al. (2015) Linking Early Geospatial Documents, One Place at a Time: Annotation of Geographic Documents with Recogito. *e-Perimetron*. 10 (2), 49–59. Available from: http://oro.open.ac.uk/id/eprint/43613 (Accessed 5 January 2019).

Sweigart, A. (2017) 退屈なことはPythonにやらせよう：ノンプログラマーにもできる自動化処理プログラミング. 東京: オライリージャパン.

TEI Consortium, eds. (2018) *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. Version 3.4.0. Last updated on 23 July 2018. TEI Consortium. http://www.tei-c.org/Guidelines/P5/ (Accessed 10 January 2019).

Wittern, C. et al. (2009) The making of TEI P5. *Literary and Linguistic Computing*. [Online] 24 (3), 281–296. Available from: https://academic.oup.com/dsh/article/24/3/281/968658 (Accessed 10 January 2019).

柳井孝介・庄司美沙 (2019) Pythonで動かして学ぶ自然言語処理入門. 東京: 翔泳社.