



Title	Essential Material for Humanities Researchers! Envisioning a New Path in Humanities Research with Text Data and TEI
Author(s)	Vogatza, Dimitra; Voulgaris, Nikolaos; Yoshiga, Natsuko et al.
Citation	
Version Type	VoR
URL	https://doi.org/10.18910/104315
rights	This article is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

<https://ir.library.osaka-u.ac.jp/>

The University of Osaka

TEI Hands-on

Text analysis with Run, Melos! (走れメロス)

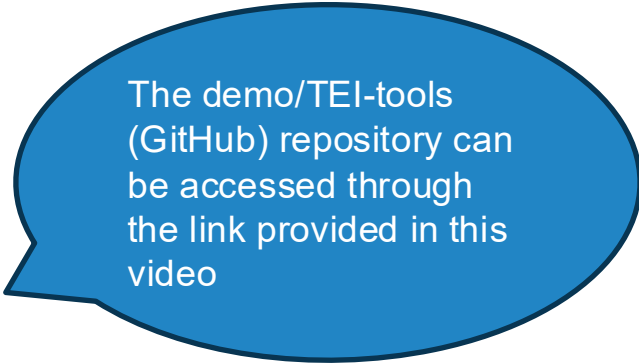
Regarding this Section

Building on the knowledge from the TEI Hands-on: Introductory Edition, we will introduce several examples of analyses using TEI-formatted research texts.

Items to prepare:

- demo(TEI-tools) > **TEI_hands-on-EN.ipynb**
- Google Chrome browser (recommended)

Google Colab <https://colab.research.google.com/>



The demo/TEI-tools (GitHub) repository can be accessed through the link provided in this video

Google Colab's Appearance

<https://colab.research.google.com/>

The screenshot displays the Google Colab web interface. At the top, the browser address bar shows 'colab.research.google.com'. Below the browser, the Colab header includes the logo, 'Welcome To Colab', and navigation links: File, Edit, View, Insert, Runtime, Tools, and Help. On the right of the header are icons for settings, a 'Share' button, a 'Gemini' icon, and a user profile picture. A secondary navigation bar contains 'Commands', '+ Code', '+ Text', 'Run all', 'Copy to Drive', 'Connect', and a collapse icon. The left sidebar, titled 'Table of contents', lists sections: 'Welcome to Colab!', 'Getting started', 'Data science', 'Machine learning', 'More Resources', and 'Featured examples', with a '+ Section' link at the bottom. The main content area is titled 'Welcome to Colab!' and features a section 'Access Popular LLMs via Google-Colab-AI Without an API Key'. This section explains that users with paid plans have access to LLMs via the 'google-colab-ai' Python library and provides a link to 'getting started with google colab ai'. Below this is a code block with the following Python code:

```
from google.colab import ai
response = ai.generate_text("What is the capital of France?")
print(response)
```

 Further down, there is a section 'Explore the Gemini API' which states that the Gemini API provides access to models created by Google DeepMind and can reason across text, images, code, and audio. It includes a 'How to get started?' section with three bullet points: 'Go to [Google AI Studio](#) and log in with your Google account.', 'Create an [API key](#).', and 'Use a quickstart for [Python](#), or call the REST API using [curl](#).' The final section, 'Discover Gemini's advanced capabilities', lists four bullet points: 'Play with Gemini [multimodal outputs](#), mixing text and images in an iterative way.', 'Discover the [multimodal Live API](#) (demo [here](#)).', 'Learn how to [analyze images and detect items in your pictures](#) using Gemini (bonus, there's a [3D version](#) as well!).', and 'Unlock the power of [Gemini thinking model](#), capable of solving complex task with its inner thoughts.'

Google Colab (Colaboratory)

A simple programming tool that runs in a web browser

<https://colab.research.google.com/>

- No need to worry about differences in OS (like macOS or Windows) or computer performance
- Allows you to write explanatory text before and after code, or create outlines of topics
- Makes it easy to understand how to write and run code
- You need to be connected to the internet and have a Google account
- To run programs, you connect to Google's computers, but the connection cannot be maintained for long periods
- If the connection is cut, any libraries or data you temporarily installed will be lost

Creating a Google Account

- To use the Google Colab service, you'll need a Google account (a combination of username and password).
- If you're just creating an account and running programs on Colab, you can use it for free.
- If you don't have an account yet, please create one before continuing.
- You can do this at:
<https://www.google.com/intl/account/about/>
- (Alternatively, try searching online for terms like "Create Google account.")

Let's Log in to Google Colab

Welcome To Colab

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all Copy to Drive

Table of contents

- Welcome to Colab!
- Getting started
- Data science
- Machine learning
- More Resources
- Featured examples
- + Section

https://colab.research.google.com

Notebook Page

Welcome to Colab!

Access Popular LLMs via Google-Colab-AI Without an API Key

Users with Colab's paid plans have free access to most popular LLMs via google-colab-ai Python library. For more details, refer to the [getting started with google colab ai](#).

```
from google.colab import ai
response = ai.generate_text("What is the capital of France?")
print(response)
```

Explore the Gemini API

Click here to Log in

Let's Open a Notebook

File > Upload notebook > Upload
select the downloaded file “**TEI_hands-on-EN.ipynb**”
You can download this file from CLE along with other sample files.

Open notebook

Examples >

Recent >

Google Drive >

GitHub >

Upload >

2

Browse

or drag a file here

The screenshot shows the Google Colab interface. The 'File' menu is open, and the 'Upload notebook' option is highlighted. A red arrow points from the 'Upload notebook' option in the menu to the 'Upload' button in the sidebar. The sidebar also shows the 'Upload' button highlighted. The main area displays a 'Welcome to Colab!' message and a code snippet for using the Gemini API.

Colab

File Edit View Insert Runtime Tools Help

Search Command

New notebook in Drive

Open notebook ⌘/Ctrl+O

Upload notebook

Rename

Save a copy in Drive

Save a copy as a GitHub Gist

Save a copy in GitHub

Save ⌘/Ctrl+S

Revision history

Download

Print ⌘/Ctrl+P

Copy to Drive

Welcome to Colab!

Access Popular LLMs via

Users with Colab's paid plans have more details, refer to the [getting started](#)

```
from google.colab import ai
response = ai.generate_text(
    prompt="Hello, my name is Gemini",
    print(response)
```

Explore the Gemini API

The Gemini API gives you access to the Gemini model from the ground up to be multimodal

TEI_hands-on-EN.ipynb

Here is a sample code with explanations that can be run directly in Google Colab:

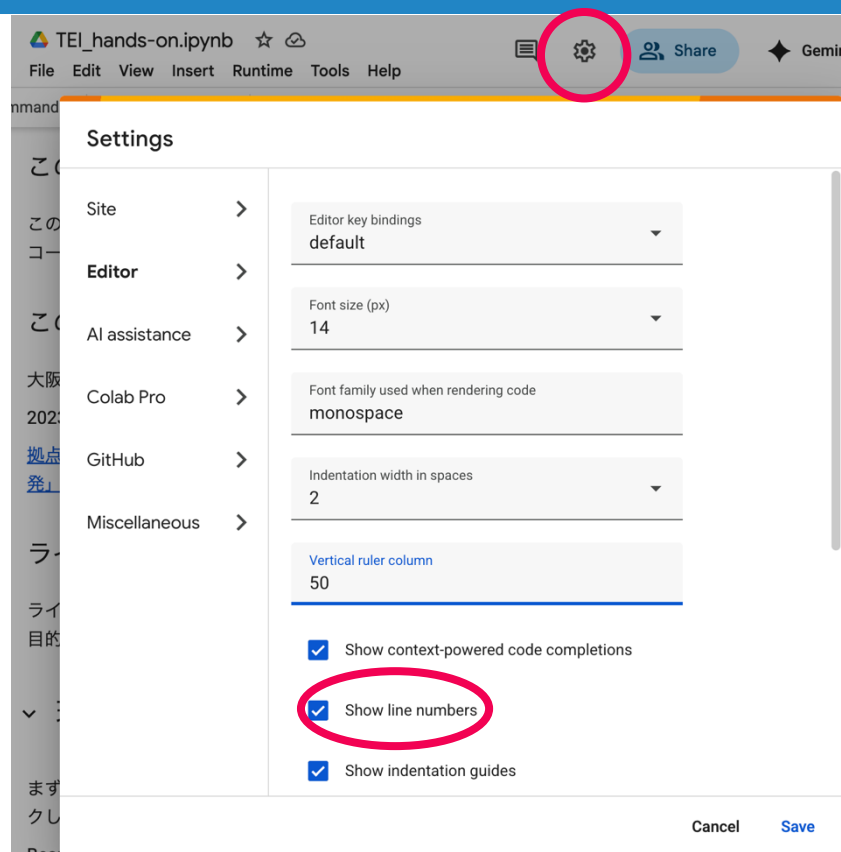
<https://colab.research.google.com/drive/19ell4FMv1FAIU3AKctemFpzo1sc57-gd?usp=sharing> (in English)

If you want to modify the code yourself, make sure to save a copy first.

The sample code is written in Python, a language with many available examples.

Google Colab Settings

To make it easier to explain using line numbers, click the gear icon for settings, then click “Editor,” and check “Show line numbers.”



Osamu Dazai's: Run, Melos! - meros.xml

A lightly modified sample based on the Aozora Bunko text, converted into TEI format.



Setting up an Environment & Loading Data

BeautifulSoup: One of the libraries (a set of bundled programs) in Python for easily parsing XML files. You can extract data by specifying tags of your choice.



```
!pip install beautifulsoup4
```

Clicking the run button executes the command



```
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages  
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages
```

Note: Make sure to click on the next cell for it to run.

Downloading meros.xml

Next, we download the TEI data from a repository called GitHub.

Click→



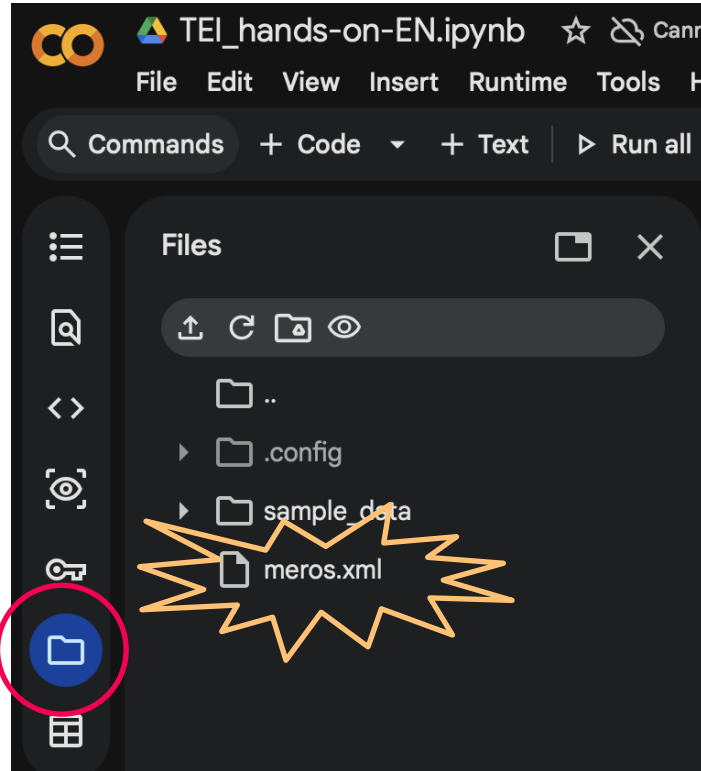
```
!curl -O 'https://raw.githubusercontent.com/nikolito/TEI-tools/main/meros.xml'
```



% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100 87937	100 87937	0 0	277k 0	--:--:--	--:--:--	--:--:--	277k

Where is the downloaded file?

Click on this
folder icon



Contents of meros.xml

A commonly used programming pattern in Python is file input and output.

```
with open("meros.xml",
encoding='utf8') as f:
print(f.read())
```



```
with open("meros.xml", encoding='utf8') as f:
    print(f.read())
```



```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title>走れメロス</title>
        <author>太宰治</author>
        <respStmt>
          <resp>Transcription</resp>
          <name>金川一之</name>
        </respStmt>
        <respStmt>
          <resp>Proofreading</resp>
          <name>高橋美奈子</name>
        </respStmt>
        <respStmt>
          <resp>TEI encoding</resp>
          <name>永崎研宣</name>
        </respStmt>
      </titleStmt>
      <publicationStmt>
        <distributor>青空文庫</distributor>
        <authority>金川一之</authority>
        <authority>高橋美奈子</authority>
        <date when="2011-01-17"> 2011年1月17日</date>
      </publicationStmt>
      <sourceDesc>
        <bibl>
          <author>太宰治</author>
          <title>走れメロス</title>
```

Toggle the display of output results

Toggle the
output on/off
from this
symbol→



Displaying text with TEI tags removed (part one)

- Strings such as `file_path` and `soup` are **variables**.
 - A **variable** is a name that refers to a **value** or object in a program
- The **value** on the right side of the equals sign is **assigned** to that name.
- A **function** takes one or more values as input and may **return a result**.

```
2  # module installation
   from bs4 import BeautifulSoup

   # Load the TEI file and have BeautifulSoup extract the XML elements.
5  file_path = "meros.xml"
   Variable

8  with open(file_path, "r", encoding="utf-8") as file:
   | soup = BeautifulSoup(file, "xml")
   Function

   # Remove the <rt> tags.
   for ruby in soup.find_all("rt"):
   |     ruby.decompose()

   # Extract the <text> element and remove all tags inside it to obtain plain text.
15 text_content = soup.find("text").get_text(strip=True)

   # Display the first 1,000 characters of the text without tags.
   print(text_content[:1000] + "...")
```

Displaying text with TEI tags removed (part two)

- A function that extracts all `rt` elements, which are tags used to display ruby text: `find_all("rt")`

An example of ruby annotation:

```
<ruby>
  <rb>邪智暴虐</rb>
  <rt>じゃちぼうぎゃく</rt>
</ruby>
```

Opening sentence: "Melos was enraged."

0 1 2 3 4 5 6 7 8

The text data stored in a variable is numbered starting from zero.

```
# module installation
from bs4 import BeautifulSoup

# Load the TEI file and have BeautifulSoup extract the XML elements.
file_path = "meros.xml"

with open(file_path, "r", encoding="utf-8") as file:
    soup = BeautifulSoup(file, "xml")

# Remove the <rt> tags.
for ruby in soup.find_all("rt"):
    ruby.decompose()

# Extract the <text> element and remove all tags inside it to obtain plain text.
text_content = soup.find("text").get_text(strip=True)

# Display the first 1,000 characters of the text without tags.
print(text_content[:1000] + "...")
```

Use the textwrap library to format the text in a more readable way

```
# Wrap the text every 20 characters and display up to 300 characters
import textwrap

wrapped_text = textwrap.fill(text_content, width=20)
print(wrapped_text[:300] + "...")
```

Slice notation

wrapped_text contains the string from index 0 to index 299, which is 300 characters

Use the **fill function** from the textwrap library to wrap text_content to a width of 20 characters, up to 300 characters.



メロスは激怒した。必ず、かの邪智暴虐の王を除かなければならぬと決意した。メロスには政治がわからぬ。メロスは、村の牧人である。

笛を吹き、羊と遊んで暮して来た。けれども邪悪に対しては、人一倍に敏感であった。きょう未明メロスは村を出発し、野を越え山越え、十里はなれた此のシラクスの市にやって来た。メロスには父も、母も無い。女房も無い。十六の、内気な妹と二人暮しだ。この妹は、村の或る律気な一牧人を、近々、花婿として迎える事になっていた。結婚式も間近かなのである。メロスは、それゆえ、花嫁の衣裳やら祝宴の御馳走やらを買いに、はるばる市にやって来たのだ。先ず、その品々を買い集...

Checking the markup rules of this file

- Which tags are used?
- How are the tags structured hierarchically?

```
from bs4 import BeautifulSoup
from IPython.display import display, HTML

def generate_collapsible_html(tag, indent=0):
    # Recursively display the XML tags in a hierarchical
    # structure
    children = tag.find_all(recursive=False)
    if not children:
        return f'<div style="margin-left: {indent}px;">
            {tag.name}</div>'

    inner_html = "".join([generate_collapsible_html(child,
        indent + 20) for child in children])
    return f'<div style="margin-left: {indent}px;">
        <summary>{tag.name}</summary>
        {inner_html}
    </div>'

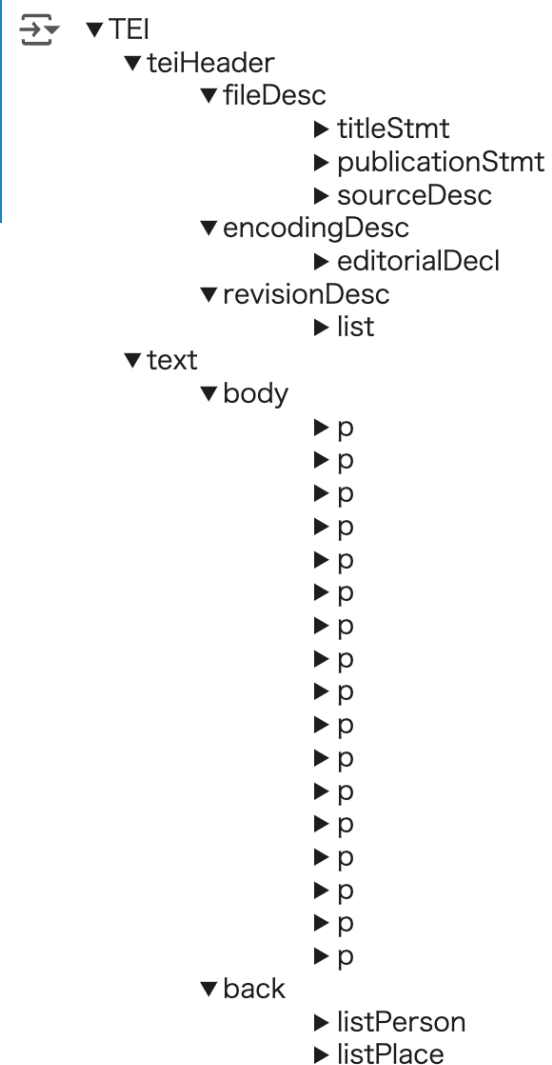
# Put the XML elements into soup
file_path = "meros.xml"
with open(file_path, "r", encoding="utf-8") as file:
    soup = BeautifulSoup(file, "xml")

# Use an HTML view to expand and collapse the tag hierarchy
root = soup.find(True) # Get the first tag
html_output = generate_collapsible_html(root)
display(HTML(html_output))
```

The tag hierarchy of meros.xml

Data in XML format is always structured as nested elements.

- As explained so far, the TEI tag contains the `teiHeader` and `text` tags.
- The `teiHeader` includes the `fileDesc`, `encodingDesc`, and `revisionDesc` tags.
- The `text` tag contains the `body`, which is the main text, and the `back`, which includes elements other than the main text, such as supplementary material.
 - The `body` contains seventeen `p` tags.
 - The `back` contains the `listPerson` and `listPlace` tags, which store the names of characters and place names that appear in the story.



Examining the editorialDecl tag

What is editorialDecl?

A tag that describes markup rules

Results output

1. In the type attribute of **placeName**, locations that actually exist are classified as “real,” and locations that do not exist are classified as “unreal.”
2. The **persName** tag is assigned to nouns that refer to people or groups of people, excluding personal pronouns.
3. Personal pronouns are marked with the **rs** tag.
4. All of the tags mentioned above use the **@corresp** attribute to reference IDs.

```
# module installation
from bs4 import BeautifulSoup
import textwrap

# Load the TEI file and let BeautifulSoup extract
the XML elements
file_path = "meros.xml"
with open(file_path, "r", encoding="utf-8") as
file:
    soup = BeautifulSoup(file, "xml")

# Select the <editorialDecl> element, which
contains the markup rules,
# and assign it to the variable editorialDecl
editorialDecl = soup.select('editorialDecl')

# Output the contents of editorialDecl
wrapped_text = textwrap.fill(editorialDecl[0].
get_text(strip=True), width=50) # Feel free to
adjust the width using the width parameter
print(wrapped_text.replace(" ", ""))
```

Let's normalize names

```
from bs4 import BeautifulSoup
from collections import Counter

# Load the TEI file and parse it with BeautifulSoup
file_path = "meros.xml" # 実際のパスを指定
with open(file_path, "r", encoding="utf-8") as file:
    soup = BeautifulSoup(file, "xml")

# Extract the <persName> elements whose corresp="#メロス"
# attribute matches the target
melos_names = [pers.text for pers in soup.find_all(
    ("persName", {"corresp": "#メロス"})

# Count the different names and their frequencies
name_counts = Counter(melos_names)

# Display the results
for name, count in name_counts.items():
    print(f"{name}: {count} times")
```

Output→

Problematic
sections

```
メロス: 68 times
メロス: 6 times
下賤
げせん
の者: 1 times
嘘つき: 1 times
兄: 4 times
メロス ほどの男: 1 times
おまえの兄: 1 times
たぶん偉い男: 1 times
真の勇者、メロス: 1 times
自分: 1 times
よくよく不幸な男: 1 times
裏切者: 1 times
地上で最も、不名誉の人種: 1 times
悪徳者: 1 times
醜い裏切り者: 1 times
正義の士: 1 times
正直な男: 2 times
勇者: 1 times
```

Let's normalize names (improved version)

```
from bs4 import BeautifulSoup
from collections import Counter

# Load the TEI file and parse it with BeautifulSoup
file_path = "meros.xml"
with open(file_path, "r", encoding="utf-8") as file:
    soup = BeautifulSoup(file, "xml")

# Extract the elements among the <persName> tags that
# correspond to corresp="#メロス"
meros_names = []

for pers in soup.find_all("persName", {"corresp": "#メロス"}):
    # If an <rb> tag is present, extract its contents; if
    # not, use the full text of the element
    if pers.find("rb"):
        meros_names.append(''.join(rb.text for rb in pers.
            find_all("rb")).strip())
    else:
        meros_names.append(pers.text.strip())

# Count the different name variants and their number of
# occurrences
name_counts = Counter(meros_names)

# Display the results
for name, count in name_counts.items():
    print(f"{name}: {count} times")
```

Example of ruby annotation:

```
<ruby>
  <rb>邪智暴虐</rb>
  <rt>じゃちぼうぎゃく</rt>
</ruby>
```

```
メロス: 74 times
下賤: 1 times
嘘つき: 1 times
兄: 4 times
メロス ほどの男: 1 times
おまえの兄: 1 times
たぶん偉い男: 1 times
真の勇者、メロス: 1 times
自分: 1 times
よくよく不幸な男: 1 times
裏切者: 1 times
地上で最も、不名誉の人種: 1 times
悪徳者: 1 times
醜い裏切り者: 1 times
正義の士: 1 times
正直な男: 2 times
勇者: 1 times
```

Let's observe how Melos is referred to in each paragraph

```
from bs4 import BeautifulSoup
from collections import defaultdict
from IPython.display import display, Markdown

# Load the TEI file and parse it with BeautifulSoup
file_path = "meros.xml" # Actual Path
with open(file_path, "r", encoding="utf-8") as file:
    soup = BeautifulSoup(file, "xml")

# For the text > body > p tags, extract the ways Melos
# is referred to within each <p> tag
paragraphs = soup.find("text").find("body").find_all("p")
melos_names_by_paragraph = defaultdict(list)

for i, p in enumerate(paragraphs, start=1):
    for pers in p.find_all("persName", {"corresp": "#メロス"}):
        if pers.find("rb"):
            name = ''.join(rb.text for rb in pers.find_all("rb")).strip()
        else:
            name = pers.text.strip()
        melos_names_by_paragraph[i].append(name)

# Display the results
for para_num, names in melos_names_by_paragraph.items():
    display(Markdown(f"### Paragraph {para_num}"))
    display(Markdown(", ".join(names) if names else "__"))
```

Paragraph 1

メロス, メロス, メロス

Paragraph 2

メロス, メロス, メロス

Paragraph 3

メロス

Paragraph 4

メロス, メロス, メロス, メロス

Paragraph 5

メロス, メロス, メロス, メロス, メロス, メロス, 下
賤, メロス, メロス, 嘘つき, メロス

Paragraph 6

メロス, メロス, メロス

Paragraph 7

メロス, メロス, 兄, 兄, 兄, メロス, メロス, メロス,
メロス

Paragraph 8

メロス, メロス, メロス, メロス ほどの男, おまえの
兄, 兄, たぶん偉い男, メロス, メロス, メロス

Paragraph 9

メロス, メロス

Paragraph 10

メロス, メロス, メロス

Paragraph 11

メロス, メロス, メロス, メロス, メロス, メロス

Paragraph 12

メロス

Paragraph 13

メロス, メロス, 真の勇者、メロス, 自分, よくよく
不幸な男, 裏切者, 地上で最も、不名誉の人種, 悪
徳者, 醜い裏切り者

Paragraph 14

メロス, メロス, メロス, 正義の士, 正直な男, 正直な
男

Paragraph 15

メロス, メロス, メロス

Paragraph 16

メロス, メロス, メロス, メロス, メロス, メロス, メ
ロス

Paragraph 17

メロス, メロス, メロス, メロス, メロス, メロス, メ
ロス, メロス, メロス, メロス, メロス, メロス, 勇者

Extracting the names of Melos, Dionysius, and Selinuntius by paragraph

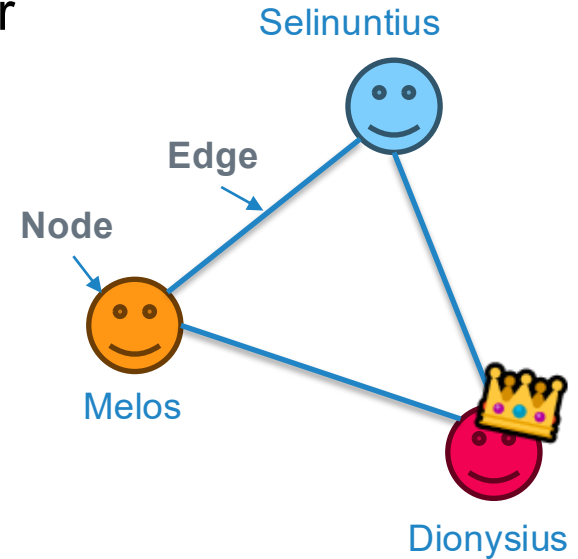
index	メロス	ディオニス	セリヌンティウス	Text (first 100 chars)
P1	メロス、メロス、メロス	邪智暴虐		メロスは激怒した。必ず、かの邪智暴虐じゃちぼうぎやくの王を除かなければならぬと決意した。メロスには政治がわからぬ。メロスは、村の牧人である。笛を吹き、
P2	メロス、メロス、メロス			きょう未明メロスは村を出発し、野を越え山越え、十里はなれた此このシラクスの市にやって来た。メロスには父も、母も無い。女房も無い。十六の、内気な妹と二人暮した。この妹は、村の或る律気な一牧人を、近々、
P3	メロス		竹馬の友、セリヌンティウス	メロスには竹馬の友があった。セリヌンティウスである。今は此のシラクスの市で、石工をしている。その友を、これから訪ねてみるつもりなのだ。久しく逢わなかつ
P4	メロス、メロス、メロス、メロス	王、王様、御自身、国王、王		歩いているうちにメロスは、まちの様子を怪しく思った。ひっそりしている。もう既に日も落ちて、まちの暗いのは当りまえだが、けれども、 なんだか、夜のせいばかりでは無く、市全体が、やけに寂しい。のんきなメ
P5	メロス、メロス、メロス、メロス、メロス、メロス、下賤、メロス、メロス、嘘つき、メロス	王、ディオニス、王、暴君、王、王、暴君、王、王、暴君、わし	セリヌンティウス、無二の友人、あの友人、身代りの男、身代りの男、身代り、その身代り	メロスは、単純な男であった。買い物、背負ったままで、 のそのそ王城にはいって行った。たちまち彼は、巡邏じゅんらの警吏に捕縛された。調べられて、メロスの懐中からは 短剣が出て来たので、

Co-occurrence network analysis of characters

Analyzing how often specific characters appear together (co-occurrence), paragraph by paragraph.

Steps :

1. Extract the characters that appear in each paragraph using the persName tag
2. Count pairs of character IDs that appear together in each paragraph
3. Create a network graph and represent co-occurrence frequency by the thickness of the edges



A **network graph** is data that represents relationships using points, called **nodes**, and lines, called **edges**

Preparing for character co-occurrence network analysis

Preparation

Load the required libraries

Necessary Libraries

- **NetworkX**: This is a powerful Python library for working with network graph structures. It is designed to create, manipulate, analyze, and visualize networks composed of nodes, which are points, and edges, which are lines.
- **matplotlib**: This is a representative Python library for data visualization. It is particularly strong in creating two-dimensional graphs and can draw a wide range of visualizations, from simple line graphs to complex heatmaps and three-dimensional plots.

Extracting the characters that appear in the story

```
# Load the TEI file and parse it with BeautifulSoup
file_path = "meros.xml" # Actual path location
with open(file_path, "r", encoding="utf-8") as file:
    soup = BeautifulSoup(file, "xml")

# Extract character IDs and names from <listPerson>
all_characters = {}

8 for person in soup.find_all("person"):
    person_id = person.get("xml:id")
    pers_name = person.find("persName")
    if person_id and pers_name:
        name = ''.join(rb.text for rb in pers_name.
            find_all("rb")).strip() if pers_name.find("rb")
        else pers_name.text.strip()
        all_characters[person_id] = name

for character in all_characters:
    print(f"{character}: {all_characters[character]}")
```

(meros.xmlより)

<listPerson>

<person xml:id="メロス">

<persName>メロス</persName>

<occupation>牧人</occupation>

</person>

メロス: メロス
ディオニス: ディオニス
セリヌンティウス: セリヌンティウス
メロスの妹: メロスの妹
メロスの妹の婿: メロスの妹の婿
ディオニスの妹婿: 妹婿さま
ディオニスの妹: 妹さま
ディオニスの妹の御子: 御子さま
ディオニスの世嗣: お世嗣
アレキス: アレキス
老爺: 老爺
フィロストラトス: フィロストラトス

Name normalization using the ID in the corresp attribute

```
# Collect co-occurrence data for each paragraph
paragraphs = soup.find("text").find("body").find_all("p")
cooccurrence = defaultdict(list)

for para_num, p in enumerate(paragraphs, start=1):
    present_characters = set()

    for pers in p.find_all("persName"):
        corresp = pers.get("corresp")
        if corresp is not None:
            corresp = corresp.strip("#")
            if corresp in all_characters:
                present_characters.add(corresp)

    # Record co-occurrences for each paragraph.
    for char1 in present_characters:
        for char2 in present_characters:
            if char1 != char2:
                cooccurrence[(char1, char2)].append(
                    para_num)

print(f"Number of co-occurring pairs: {len(cooccurrence)}")
print("Co-occurring pairs : Paragraph numbers in which they co-occurred")
for occurrence in cooccurrence:
    print(f"{occurrence}: {cooccurrence[occurrence]}")
```

共起ペアの数: 78

共起ペア : 共起したパラグラフ番号

('メロス', 'ディオニス'): [1, 4, 5, 6, 8, 9, 10, 12, 13, 16, 17]

('ディオニス', 'メロス'): [1, 4, 5, 6, 8, 9, 10, 12, 13, 16, 17]

('メロス', 'メロスの妹の婿'): [2, 7, 8]

('メロス', 'メロスの妹'): [2, 5, 7, 8]

('メロスの妹の婿', 'メロス'): [2, 7, 8]

('メロスの妹の婿', 'メロスの妹'): [2, 7, 8]

('メロスの妹', 'メロス'): [2, 5, 7, 8]

('メロスの妹', 'メロスの妹の婿'): [2, 7, 8]

('メロス', 'セリヌンティウス'): [3, 5, 6, 10, 11, 13, 15, 16, 17]

('セリヌンティウス', 'メロス'): [3, 5, 6, 10, 11, 13, 15, 16, 17]

('メロス', 'ディオニスの世嗣'): [4]

('メロス', 'ディオニスの妹'): [4]

('メロス', 'ディオニスの妹の御子'): [4]

('メロス', 'アレキス'): [4]

('メロス', 'ディオニスの妹の婿'): [4]

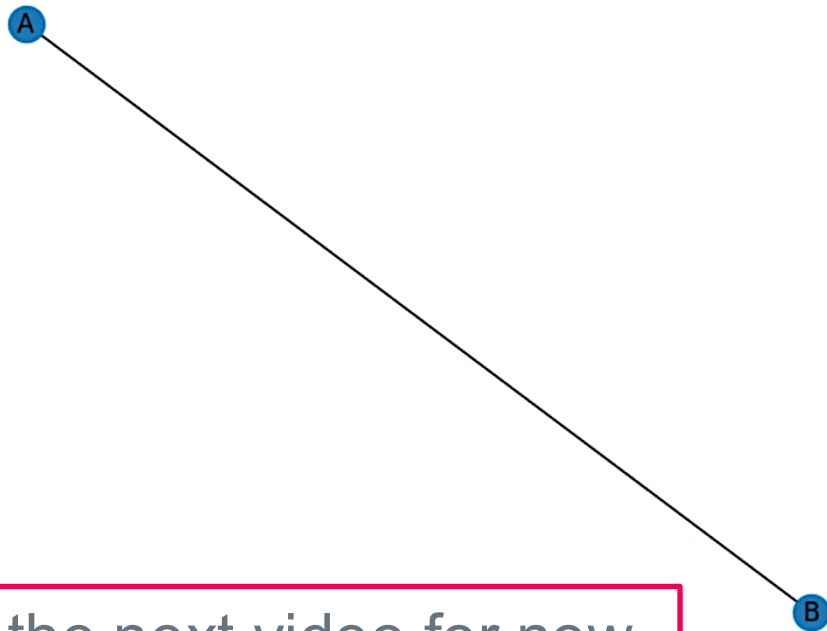
By checking the corresp attribute, we can identify whose name it is by its ID.

Fundamentals of NetworkX

Please refer directly to the sections from “What is NetworkX?” to “Code explanation” in the Google Colab notebook “TEI_hands-on-EN.ipynb”.

These sections explain the basics of drawing edges between nodes.

```
Nodes: ['A', 'B']  
Edges: [('A', 'B', {'weight': 3})]
```



It's okay to move on to the next video for now.

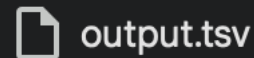
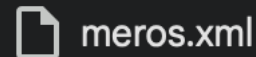
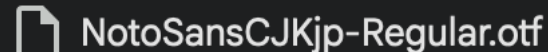
Visualizing character co-occurrence by paragraph using NetworkX (1)

```
import networkx as nx
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup
from collections import defaultdict
import matplotlib.font_manager as fm
import matplotlib.patches as mpatches
```

```
# Japanese font configuration (use an appropriate font
when running in Google Colab)
# In Colab, fonts such as IPAexGothic are available
! curl -O "https://raw.githubusercontent.com/notofonts/noto-cjk/main/Sans/OTF/Japanese/NotoSansCJKjp-Regular.otf"
```

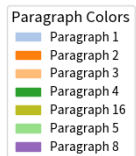
```
font_path = 'NotoSansCJKjp-Regular.otf'
font_prop = fm.FontProperties(fname=font_path)
fm.fontManager.addfont(font_path)
plt.rcParams['font.family'] = font_prop.get_name()
```

Files

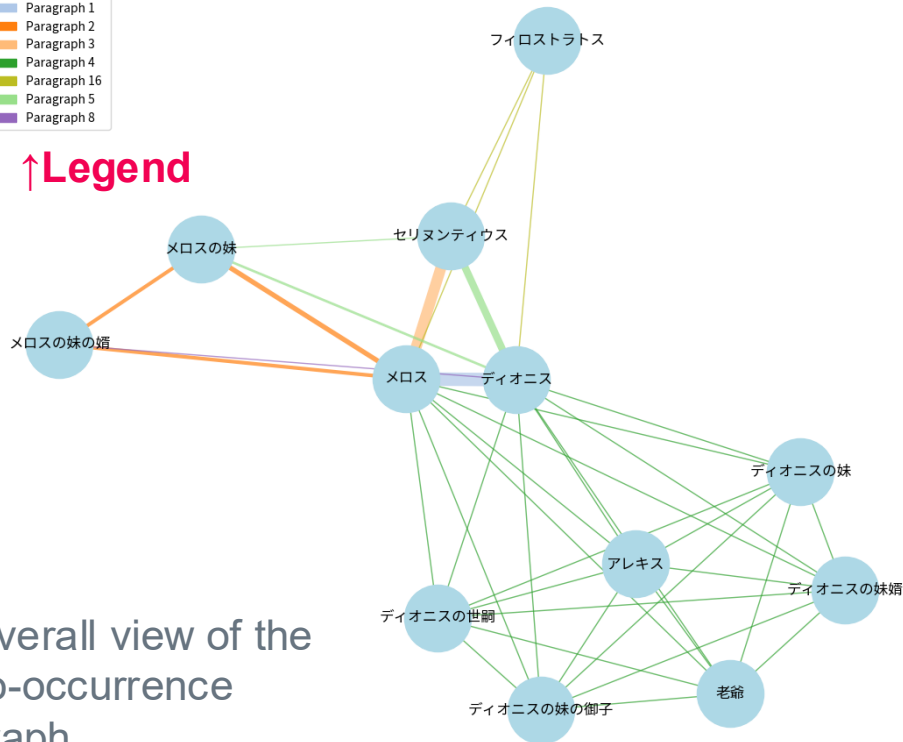


Visualizing character co-occurrence by paragraph using NetworkX (2)

登場人物の共起ネットワーク (パラグラフ別色分け + エッジ太さは共起数)



↑ Legend



Overall view of the co-occurrence graph

How to read this graph:

Edges between characters

1. thickness
2. number
3. color variety

These reveal relationship importance, character importance, and changes in relationships over the course of the story.

For deeper analysis, compare the network with the content of each paragraph.

Summary

- An introductory text analysis using Run, Melos
- The notebook can be copied and adapted to TEI files of other works