# Language Teaching with Drama Plays

Felipe Nobrega

## Table of contents

## Language Teaching with Drama Plays - Report

### Felipe Nobrega - Text Technology Project Description

*Goal*

Provide easy access to grammar topics that are present on William Shakespeare drama plays and may be useful for language teachers. Therefore, the project can be used to facilitate the usage of literature in the classroom and shorten class preparation time.

*How?*

- Website that allows queries for specific grammar topics within the corpus of drama plays.
- Provide a list with results of the query and the context of the drama play where it is found.

This project is based in three steps according to the project requirements of the course. Below we can find an overview of the project along with the required steps.

| Step | Description |
|---|---|
| Collect | 1. Collect data from Shakespeare plays. DraCor Corpora was used. The files are found in .xml. |
| Prepare | 1. Perform data extraction from .xml using XPath in XML Copy Editor. Initial goal: Modal verbs in Romeo and Juliet.<br>2. Parse the extracted data in simple text to be processed by PostgreSQL in order to populate tables.<br>3. Create a database for the plays and tables for grammar topics. |
| Access | 1. Create a Python app to connect to PostgreSQL to get information and perform queries within the plays database.<br>2. Display the results in a webpage - a table with the line, section, desired word and context from the play. |

## Collect

### DraCor Drama Corpora

Access https://dracor.org/shake and download the .xml files corresponding to the following plays:

| Play | Processed | Database | Webpage |
|---|---|---|---|
| Romeo and Juliet | OK | OK | OK |
| Hamlet | No | No | No |
| Macbeth | No | No | No |
| King Lear | No | No | No |
| Othello | No | No | No |

## Prepare

### XPath commands

1. Extraction of *lemma* **can** from the play:

```
//w[@lemma='can']
```

## Python commands

1. Solve the problem of special character at the beginning of the plays. In order to remove the special character that prevented the **XMLCopy Editor** from doing **XPath**, I have created the following script to automatically replace the first line of every XML file.

```python
import os

def replace_first_line(file_path):
    with open(file_path, 'r', encoding='utf-8-sig') as file:
        lines = file.readlines()

    # Check if the first line starts with the specified string
    if lines and lines[0].startswith('<TEI xmlns='):
        lines[0] = '<TEI>\n'

    # Write the modified content back to the file
    with open(file_path, 'w', encoding='utf-8') as file:
        file.writelines(lines)

    print(f"Successfully processed: {file_path}")

def process_xml_files(directory):
    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith('.xml'):
                file_path = os.path.join(root, file)
                replace_first_line(file_path)

# Specify the directory containing your XML files
directory = r'D:\lang_drama_plays\plays'
process_xml_files(directory)
```

2. Parse XML in order to extract information in formatted text in order to feed Postgres tables.

```python
import xml.etree.ElementTree as ET

# Parse XML file
tree = ET.parse('modal_verbs_romeo_juliet.xml')
root = tree.getroot()

```

```
# Iterate through each 'w' element in the XML
for w in root.findall('w'):
    id = w.get('id')
    section = w.get('n')
# Extract element value
    element_value = w.text.strip() if w.text else ''

# Format the output
    formatted_output = f"('{id}', '{section}', '{element_value}'),"

# Print the results
    print(formatted_output)
```

3. XML to HTML converter. Convert extracted original text from XML to later link *text_id* element to *span class* in HTML format.

```
from lxml import etree

def convert_xml_to_simple_html(xml_file, html_file):
    tree = etree.parse(xml_file)
    root = tree.getroot()

    html = etree.Element("html")
    head = etree.SubElement(html, "head")
    etree.SubElement(head, "meta", charset="utf-8")
    etree.SubElement(head, "title").text = "Romeo and Juliet"
    style = etree.SubElement(head, "style")
    style.text = "body { line-height: 1.5; } .highlight { background-color: yellow; }"
    body = etree.SubElement(html, "body")
    text_div = etree.SubElement(body, "div", id="text")

    current_section = None
    for w in root.findall('.//w'):
        if w.get('n') and w.get('n') != current_section:
            current_section = w.get('n')
            section_span = etree.SubElement(text_div, "span", id=w.get('id'), class_=current_
            section_span.text = f"\n\n[{current_section}]\n"

        word_span = etree.SubElement(text_div, "span", id=w.get('id'))
        word_span.text = w.text
        word_span.tail = " "
    script = etree.SubElement(body, "script")
    script.text = """
    const wordId = window.location.hash.slice(1);
    if (wordId) {
        const word = document.getElementById(wordId);
        if (word) {
            word.classList.add('highlight');
            word.scrollIntoView({ behavior: 'smooth', block: 'center' });
        }
    }
    """

    with open(html_file, 'wb') as f:
        f.write(etree.tostring(html, pretty_print=True, method="html", encoding="utf-8"))

# Usage
convert_xml_to_simple_html('romeo_juliet_text.xml', 'romeo_and_juliet.html')
```

## Database Commands

### sql commands

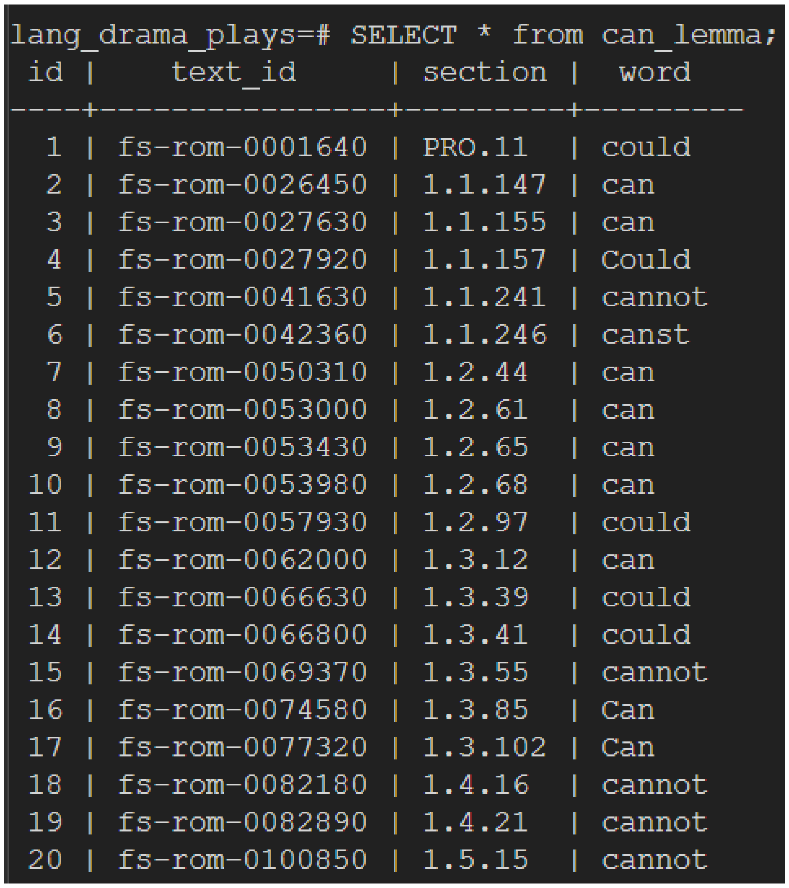
1. Create Tables Command to create tables for *Modal Verbs*

```
CREATE TABLE can_lemma (
    id SERIAL PRIMARY KEY,
    text_id TEXT NOT NULL,
    section TEXT NOT NULL,
    word TEXT NOT NULL
);
```

2. Insert extracted data from XML files

```
INSERT INTO can_lemma (text_id, section, word) VALUES
('fs-rom-0001640', 'PRO.11', 'could'),
('fs-rom-0026450', '1.1.147', 'can'),
(...)
```

*Table example*



```
lang_drama_plays=# SELECT * from can_lemma;
 id |     text_id      | section |  word
----+------------------+---------+---------
  1 | fs-rom-0001640 | PRO.11  | could
  2 | fs-rom-0026450 | 1.1.147 | can
  3 | fs-rom-0027630 | 1.1.155 | can
  4 | fs-rom-0027920 | 1.1.157 | Could
  5 | fs-rom-0041630 | 1.1.241 | cannot
  6 | fs-rom-0042360 | 1.1.246 | canst
  7 | fs-rom-0050310 | 1.2.44  | can
  8 | fs-rom-0053000 | 1.2.61  | can
  9 | fs-rom-0053430 | 1.2.65  | can
 10 | fs-rom-0053980 | 1.2.68  | can
 11 | fs-rom-0057930 | 1.2.97  | could
 12 | fs-rom-0062000 | 1.3.12  | can
 13 | fs-rom-0066630 | 1.3.39  | could
 14 | fs-rom-0066800 | 1.3.41  | could
 15 | fs-rom-0069370 | 1.3.55  | cannot
 16 | fs-rom-0074580 | 1.3.85  | Can
 17 | fs-rom-0077320 | 1.3.102 | Can
 18 | fs-rom-0082180 | 1.4.16  | cannot
 19 | fs-rom-0082890 | 1.4.21  | cannot
 20 | fs-rom-0100850 | 1.5.15  | cannot
```

## Access

### Python commands

1. Webapp created using Flask library to connect to PostgreSQL database and perform queries that will be displayed in .html webpages.

```python
from flask import Flask, render_template, request, send_from_directory
import psycopg2
import os

app = Flask(__name__)

# Connect to PostgreSQL database
def get_db_connection():
    conn = psycopg2.connect(
        dbname='lang_drama_plays',
        user='postgres',
        password='',
        host='localhost',
        port='5432'
    )
    return conn

# Define root and main page template to be displayed
@app.route('/')
def index():
    return render_template('index.html')

# Query in database
@app.route('/search', methods=['POST'])
def search():
    modal_verb = request.form['modal_verb'].lower()
    table_name = f"{modal_verb}_lemma"

    conn = get_db_connection()
    cur = conn.cursor()

    query = f"SELECT text_id, section, word FROM {table_name}"
    cur.execute(query)
    results = cur.fetchall()

    cur.close()
    conn.close()

    return render_template('results.html', results=results, modal_verb=modal_verb) #results

@app.route('/text/<filename>')
def serve_text(filename):
    return send_from_directory('text', filename)

if __name__ == '__main__':
    app.run(debug=True)
```