KJC4 4 CREDITS

NARRATIVE HOMEWORK12

# ==================================

# doing stuff you you, don't touch

from lxml import etree

infile = open('hamlet-tei.xml', 'rb')

xml = infile.read()

infile.close()

tree = etree.fromstring(xml)

ns = {'tei': 'http://www.tei-c.org/ns/1.0'}

# ==================================

# Because this has a namespace, you'll need to provide

# tei: before each of the element names (but not attributes) that you use.

# Example: '//tei:author'

# Example query that prints out the author text in the titleStmt node

# print(tree.xpath('//tei:titleStmt/tei:author/text()', namespaces = ns))

# this namespaces = ns piece will need to be in every xpath query that you run

# \*E-V-E-R-Y\* query

# you'll see that (at least at part of the beginning), I have given you

# some code with an empty xpath statement (as an empty string).

# You'll need to fill in the xpath query.

# I stopped doing this after a few, so then you're on your own.

# 1 ------

# Write a statement that selects the text from all the title elements. (7 results)

titles = tree.xpath('//tei:title/text()', namespaces = ns)

print(len(titles))

This selects the text from all the title elements using the text(), the output is stored in a variable “titles”. Then we find the length of titles

# 2 ------

# Write a statement that finds the text of all the title elements inside the titleStmt node (3 results)

q1rusul = tree.xpath('//tei:titleStmt/tei:title/text()', namespaces = ns)

print(q1rusul)

print(len(q1rusul))

This selects all the text from the title elements and stores them in a variable ”q1rusul”. We then find the length of q1rusul

# 3 ---------

# Write an xpath query that selects all the type attributes from the div elements (26 results).

# Create a Counter object (remember to import the collections module) with the results of your xpath query as a parameter

# (see the Dictionary week lecture notes for more info). Print out how many of each value appeared in the text.

# Your results should be: Counter({'scene': 20, 'act': 5, 'play': 1})

from collections import Counter

dict={}

typeact=tree.xpath('//tei:div[@type="act"]', namespaces = ns)

typescene=tree.xpath('//tei:div[@type="scene"]', namespaces = ns)

typeplay=tree.xpath('//tei:div[@type="play"]', namespaces = ns)

dict['act']=len(typeact)

dict['scene']=len(typescene)

dict['play']=len(typeplay)

print(Counter(dict))

We create an empty dictionary “dict”, We then find the type attribute values of act,scene and play for the div elements. We create key:value pairs for each of those. The values are the length of each type attribute value.

# 4 --------

# Review these results and look back at the elements in the file. What do these results tell you about how

# meaning of the div elements in the XML? (https://en.wikipedia.org/wiki/Hamlet#Plot has more info on the play if you need it)

# You can write your answer in code comments. You should have about 100 words of description and analysis.

the div element is used to group the content of the play. Asides the first one which has a type of play,

Each div element has a type attribute with text value of act,an n attribute with text value representing the act number,

there are only 5 acts but varying number of scenes in each act

then another div element contained within with type attributes that have text value of scene and an n attribute with

text values representing the scene number

# 5a ---------

# Write an xpath query that finds all the div elements that are for scenes

# (Note that you are just finding the elements here, not extracting anything out.)

# You should find 20 elements, and when printed out the results should be something but not exactly like this:

# [<Element {http://www.tei-c.org/ns/1.0}div at 0x10d216a88>, <Element {http://www.tei-c.org/ns/1.0}div at 0x10d216b48>, etc.

allscenedivs = tree.xpath('//tei:div[@type="scene"]', namespaces = ns)

print(len(allscenedivs))

This finds the type attribute values that are scene for the div elements and stores them in a variable “allscenedivs”. We then find the length of “allscenedivs”

# 5b ---------

# Now use python to loop over all those elements, and in that loop do the following:

# - add an xpath query that extracts the type attribute

# - add an xpath query that extracts the n attribute

# - recall that these results will be inside a list, so you'll need to extract the [0] element out of each

# - Create a print statement that states "Scene: #" where each # is replaced by the appropriate value

# Here are the first 4 print results:

# scene: 1

# scene: 2

# scene: 3

# scene: 4

# I've set up a base for you here. Replace the empty strings with the needed xpath query fragment.

# important point: your xpath on these elements will not need to include an opening / or //

# this is because you are operating on elements directly, so no need to declare anything about the root

# Base

for div in allscenedivs:

n = div.xpath('@n', namespaces = ns)

type\_value = div.xpath('@type', namespaces = ns)

# construct you print statement here

print(type\_value[0],":",n[0])

This finds the type attribute values that are scene for the div elements and stores them in a variable “allscenedivs”. We then loop through the elements in allscenedivs. We select the value of n attribute and store it in a variable”n”, select the value of the type attribute and store it in a variable “type\_value”. We then print out the various scenes

# 5c ---------

# Copy your 5b answer's for loop below and add the following to it for 5c

# You are again working with the div elements that are just for the scenes.

# As you previously discovered for answering item 4, the scene divs live inside of the act divs.

# Our goal here is now to print out not just the scene information, but add the act

# number in that same line.

# Inside your for loop, add the following things:

# - an xpath query that finds the parent element's type attribute value

# - an xpath query that finds the parent element's n attribute value

# - add these things into your existing print statement.

# Again, you should end up with 20 lines of text being printed out.

# Here are the first 4 results again:

# act: 1 scene: 1

# act: 1 scene: 2

# act: 1 scene: 3

# act: 1 scene: 4

for div in allscenedivs:

n = div.xpath('@n', namespaces = ns)

n2 = div.xpath('../@n', namespaces = ns)

type\_value = div.xpath('@type', namespaces = ns)

type\_value2 = div.xpath('../@type', namespaces = ns)

# construct you print statement here

print(type\_value2[0],n2[0],type\_value[0],":",n[0])

This finds the type attribute values that are scene for the div elements and stores them in a variable “allscenedivs”. We then loop through the elements in allscenedivs. We select the value of n attribute and store it in a variable”n”, Select the n attribute value of the parent node and store it in “n2”. Next we select the value of the type attribute and store it in a variable “type\_value”, select the type attribute value of the parent node and store it in “type\_value2”. We then print out the various acts, act numbers,scenes, scene numbers.

# ---------

# FOUR CREDIT HOUR STUDENTS BELOW

# ---------

# 6a ------

# Look at the structure of the person elements that appear inside the listPerson

# element just a little way down from the top of the file.

# Each contains several personal names (in the persName element) with a type attribute,

# several other kinds of personal information, and the person's social status

# Write a query that selects (but does not extract anything from) the person elements.

person\_element = tree.xpath('//tei:listPerson/tei:person', namespaces = ns)

We select the person elements inside the listPerson elements and store in a variable “person\_element”

# 6b ------

# As you've done before, you're going to loop over these element results and pull out these values

# The problem here is that not all characters have status values or occupations.

# Take a look at the results and you'll see that these queries are returning a list

# for everything, even if there's no results.

# I'm providing a function for you to use in the cases where you are expecting

# to have one result within that list. This will allow you to grab the 0th element

# when there is only one element, and provide a missing value when there is none.

# - xpathresult: the results of your xpath query, this will be a list

# - expected\_num\_results: the number of results you expect to get from

# the xpathresult

# - missing\_value: the value that you want to get back when there aren't any results

def checkFor1Result(xpathresult, missing\_value):

if len(xpathresult) > 1:

howmany = len(xpathresult)

raise ValueError("Your list had " + str(howmany) + " items instead of 1. Shutting down the program.")

elif len(xpathresult) == 1:

result = xpathresult[0] # grab the element when there is exactly one to grab

else:

result = missing\_value

return result

# For example:

# empty\_occ\_example = [] # an empty list is what you'll get when the element doesn't exist

# print(checkFor1Result(empty\_occ\_example, "MissingOccupation"))

# # this prints out: MissingOccupation

# occ\_example = ['military'] # this is what your list will look like when you find a result

# print(occ\_example)

# print(checkFor1Result(occ\_example, "MissingOccupation"))

# # this prints out: military

# Now you're ready to get to work.

# Loop over your person element results, and for each element:

# - write an xpath query that finds the id of the person, e.g. "#F-ham-mar"

# - write an xpath query that finds the socecStatus element text

# - write an xpath query that finds the occupation element text

# - run the results of these three queries through the checkFor1Result function (and add an appropriate missing value)

# - print out the results in one line, in this order: id, status, occupation

# First 3 printed results should be (with the missing messages that you use):

# F-ham-pla.1 worker MissingOccupation

# F-ham-all MissingStatus MissingOccupation

# F-ham-amb noble MissingOccupation

# Put your for loop below

for element in person\_element:

id = element.xpath('@xml:id', namespaces = ns)

status= element.xpath('./tei:socecStatus/text()', namespaces = ns)

occupation= element.xpath('./tei:occupation/text()', namespaces = ns)

# construct you print statement here

print(id[0],checkFor1Result(status, "MissingStatus"),checkFor1Result(occupation, "MissingOccupation"))

We select the person elements inside the listPerson elements and store in a variable “person\_element”. We then loop through the elements in person\_element. We select the xml:id attribute and store in “id”, socecStatus element text value and store in “status”,occupation element text value and store in “occupation”, We then print out the values of these variables, substituting “missing” where empty lists would have been produced

# Now make a copy of the for loop you just wrote and modify it to:

# - collect these new results into a list within the for loop

# - use an accumulator to collect all those lists

# this will make a list of 33 lists. Each one of those lists will have

# 3 elements. Example: Here's the first 2 lists in it:

# ['F-ham-pla.1', 'worker', 'MissingOccupation'],

# ['F-ham-all', 'MissingStatus', 'MissingOccupation']

# (again you might have different missing string values)

# put your new for loop below

allrows=[]

for element in person\_element:

id = element.xpath('@xml:id', namespaces = ns)

status= element.xpath('./tei:socecStatus/text()', namespaces = ns)

occupation= element.xpath('./tei:occupation/text()', namespaces = ns)

onerow=[id[0],checkFor1Result(status, "MissingStatus"),checkFor1Result(occupation, "MissingOccupation")]

allrows.append(onerow)

print(allrows)

Here similar to the previous questions, but first we create an empty list “allrows” and here we store each line of output as a list in a variable”onerow”, then we accumulate all these lists into “allrows”

# 7 -----

# Now, use the CSV module to write out this data.

# Let the header values be: id, status, occupation

# Let the file name be hamlet-character-data.csv

# You must use the CSV module to receive full credit

# This should yield a csv file with 3 columns and 34 rows.

#Here's a template. (Source: http://chimera.labs.oreilly.com/books/1230000000393/ch06.html#to\_write\_csv\_da)

import csv

outfile = open('hamlet\_character\_data.csv', 'w')

csv\_out = csv.writer(outfile)

csv\_out.writerow( ['id', 'status','occupation'] ) # add your

csv\_out.writerows(allrows)

outfile.close()

Here we create a csv file to write all the output to. We create three columns, “id”,”status”,”occupation”. We then write all the output in “allrows” to the csv file and close.