KJC4 4CREDITS

HOMEWORK 11

Directions:

Use this website: https://www.opensourceshakespeare.org/views/plays/play\_view.php?WorkID=hamlet&Scope= entire&pleasewait=1&msg=pl

You can use any tool that you’d like, but I’m going to suggest that you use the XPath Helper in Chrome, or google sheets. Use the spaces below to provide your XPath queries and a short narrative (a few sentences, or more if you want to say more) about your answer. I suggest that you write the narratives as you go instead of going back through after completing the work.

You’ll be provided a prompt, some sample content, and the expected number of results. Occasionally some hints as I thought of them. Answers that have a different number of results will be considered and may be given full credit if we determine that our expected answer was limited or incorrect. Use your narrative to explain why your answer is correct if there is a difference. Likewise, not all queries with the same number of results will get full points (read what it is asking for carefully). The expected results number is there to help you assess if you’re on the right track or not.

You can use that sample content to locate the item in the source HTML. Each of your queries should end with something extracting the content (review the “Extracting Stuff” section of the lecture notes if you need).

1 Write a query that finds the play title. “The Tragedy of Hamlet, Prince of Denmark”. 1 result

//h2/text()

This particular text is contained within the h2 element. We look within this element and get the text using text().

2. Write a query that finds any element (look in the lecture notes for the wildcard symbol) with a class attribute value of “playtitle”. “The Tragedy of Hamlet, Prince of Denmark”. 1 result

//\*[@class='playtitle']

As seen above, we make use of the wildcard \* which represents any element to do this. We then find any element with a class attribute value of playtitle

3. The top of the page has a table listing out all the acts. Write a query that selects all the text of those scene titles from that table. Example results: “Elsinore. A platform before the Castle.” And “Elsinore. A room of state in the Castle.” 20 results.

//td/p[@class ='playtextsmall']/a/text()

We look within the td element. We then look within the p element for the class attribute value of playtextsmall. Next we look within the a element and get the text using text()

5. Write a query that finds all the play line numbers that are visible on the right hand side of the website. These label each line by 5s. Example results: 5, 10, 15. 795 results.

//span[@class = 'playlinenum']/text()

We look within the span element for the class attribute value of playlinenum and then get the text using text(). This gets the play line numbers that are visible on the right hand side of the website.

6. Looking inside the HTML, you can see that the line number start for each character’s speech is labeled with an a element. These elements have a name attribute with the line numbers. Write a query that finds all those line numbers. Be careful! This name attribute is used elsewhere. Example results: 2, 3, 4. 1137 results.

//li/a/@name

We look within the li element. We then look within the a element and access the name attribute. This produces the line numbers

7. Each spoken line of the text (as rendered on the website) has the speaker’s name on the left and the line to be spoken on the right. The names are stored in separate elements from the text. Write a query that finds all character names. Example results: Bernardo, Francisco, Bernardo (note that these don’t have a . after them). 1137 results.

//li[@class='playtext']/strong/a/text()

We look within the li element for the class attribute value of ‘playtext’. Next we discover that the character names are all in bold so we look within the strong element. Then we look within the a element and get the text using text()

8. Write a query that finds all the speaker name elements that are Bernardo. Example results: should all be Bernardo. 19 results.

//li[@class='playtext']/strong/a[.='Bernardo']/text()

We look within the li element for the class attribute value of playtext. Next we look within the strong element. Then we look to find all the a elements that have the element text value of Bernardo. We then get the text using text()

9.Write a query that finds all of Bernardo’s lines. (you’ll need to use parent notation here). Example results: Who's there?, Long live the King! (there will be 19 nodes selected but 34 results when you select the text).

//li[@class = 'playtext']/strong/a[.='Bernardo']/../../text()

We look within the li element for the class attribute value of playtext. Next we look within the strong element . Then we find all the a elements that have the element text value of Bernardo. We then go back two layers to the parent nodes in order to get the text and do so using text()

10. Write a query that finds all the play text lines (just the spoken lines, not the names or directions). Example content: “Who's there?” and “Nay, answer me. Stand and unfold yourself.” 3984 results.

//li[@class = 'playtext']/text()

We look within the li element for the class attribute value of playtext. Then we get the text using text()

11. Write a query that finds all the passages that mention "Death" (note the capital). So you're looking for spoken passages that contain the text "Death". Each line in a passage is a separate string, but you'll select the whole passage, not only the lines that contain "Death". Use the function contains(., "Death") within a logical test on the li elements. Example results: “Heaven make thee free of it! I follow thee. I am dead, Horatio. Wretched queen, adieu!” etc etc, this passage continues for several lines. 13 results (because it will be counting the total number of strings, even though they come from just the 2 passages). Looking ahead to items 12 and 13 can also help confirm that you’ve got the right passages selected.

//li[@class ='playtext'and contains(.,'Death')]/text()

We look within the li element for the class attribute value of playtext and use the function contains(.,’Death’). We connect these Boolean checks with ‘and’. Then we get the text using text()

12. Using the previous query, find the speakers for those two lines. Remember that this will be selecting the li element itself, so you’ll need to trace down from there to get the speaker element. Results: Hamlet and Fortinbras. 2 results.

//li[@class ='playtext'and contains(.,'Death')]/strong/a/text()

We look within the li element for the class attribute value of playtext and use the function contains(.,’Death’). We connect these Boolean checks with ‘and’. Then we look within the strong element , the a element and get the text using text()

13. Adapt your previous query to look up the starting line number for those passages. Results: 3990 and 4029. 2 results.

//li[@class ='playtext'and contains(.,'Death')]/a/@name

We look within the li element for the class attribute value of playtext and use the function contains(.,’Death’). We connect these Boolean checks with ‘and’. We then look within the a element and access the name attribute. This gives us the starting line numbers