Spring 2018 Status: Looking Ok.

XPath B

Now that you have gotten a bit more comfortable with XPath queries, we're going to now explore how we can use a tool inside of Python to execute those queries. There are many XPath tools out there, and there will always be differences in how individual parsers will work. So expect to need time to acclimate yourself if you are switching to a different one. You will have to test things out and adapt accordingly.

For the purposes of our lesson here, we're going to use the XPath function within lxml's etree module. This module works well and consistently over the years, but you may find that other packages will work similarly.

I'm going to set this all up for you, so you can use this pattern without thinking too much about it. But I will try and explain some of it as we go.

The basic pattern

The basic pattern that we will be exploring here is this:

1. Read in the document
2. Parse that object (either an IO object or string depending on your pattern) into a tree object.
3. Apply you desired XPath things to that tree object.

There will be one and only one way you'll need to do this class, but there are other methods out there that you'll see.

Our pattern for class will be:

1. Read the XML file with `.read()` to read in the text as a big string.
2. Pass that string the parsing method `.fromstring()` to parse into a tree object. (don't forget that you'll have to import this module)
3. Use the `.xpath()` function on that tree object to execute xpath queries on it.

Step 0: import the lxml module

```
from lxml import etree
```

Step 1: read in the file

The `rb` read in mode is required because of the encoding issues.

```
infile =  open('YOURFILENAME.xml', 'rb')
xml = infile.read() # this will be passed to the parser
infile.close()
```

Step 2: parse into a tree object

I could call this variable name anything that I want, but we usually use `tree` as a convention to indicate that it is the entire XML tree and not a constituent node. This is using the `.fromstring()` function from the lxml/etree, which will parse string text into a tree object.

```
tree = etree.fromstring(xml)
```

Step 3: use the `.xpath()` on the tree object to execute an xpath query

We'll be talking about namespaces in a later section.

Example when there in no namespace happening:

```
results = tree.xpath('//elementwhatever/text()')
```

Example when there is a namespace schema to handle:

```
results = tree.xpath('//alias:elementwhatever/text()', namespaces={'alias': "URL fo
und in the document goes here"})
```

Don't worry, this entire lesson is about unpacking more about step 3.

Our data source

We'll be using an XML document of "Hamlet" by Shakespeare. This is located in the hamlet-tei.xml file. This is a proper XML file that uses the TEI schema. https://en.wikipedia.org/wiki/Text_Encoding_Initiative (https://en.wikipedia.org/wiki/Text_Encoding_Initiative) You will want to read this now so you can understand the basics of what's going on in this file.

The data file has its own attribution, but I grabbed it as a material from this workshop: http://tei.it.ox.ac.uk/Talks/2015-08-maynooth (http://tei.it.ox.ac.uk/Talks/2015-08-maynooth)

Take some time exploring this file to better understand the structure. There's no real need to do a full TEI tutorial for this lesson. This lesson is not meant to be a tutorial on TEI, we're just using it as example data.

This is a very brief description of the structure of the Hamlet file:

- In `teiHeader`:
  - `fileDesc` node contains information about the provenance of the file and content.
  - `profileDesc/particDesc` node contains information on the characters in the play
  - `profileDesc/settingDesc` node contains setting information for the play
- In `text`:
  - this contains nodes for each act, scene, and passage.
  - each passage is in `sp` elements, with `@who` attributes representing the standardized ID for each speaker (those IDs are defined in the `particDesc` node. The `speaker` reports out what the original text had for the speaker information, and the `l` elements have the individual lines.

There are other details that you will need to explore on you own to get a feel for things.

For now, we're going to go ahead and read in our file and prepare our tree object. You'll only need to do this once at the top of your script. After that, you'll just be using the tree object.

In [62]:
```
from lxml import etree

infile =  open('hamlet-tei.xml', 'rb') # don't forget that rb in here
xml = infile.read()
infile.close()

tree = etree.fromstring(xml)
```

namespaces

Most proper XML files have namespaces (there can be multiple) that you'll need to navigate. As this is not a metadata or TEI course, I will not provide an extended discussion on what this is.

We can see in line 4 of the document, which has the root element: `<TEI xmlns="http://www.tei-c.org/ns/1.0">`

This is saying that the elements found in this root node belong to the TEI schema, with a URL to the schema definition. This information is for the parsers that are trying to validate the schema to ensure that the XML conforms to the schema. Programs such as Oxygen XML editor will do this.

Sometimes it can be tricky to tell what the namespaces are for the elements, but you'll see this URL pop up again when we dig in and print out the elements. That is often the clue that you need for how to handle the namespaces.

The patterns from step 3 show you how to handle this, but keep on reading. Right now, I want you to focus on the larger picture and just follow along with the pattern that I'm giving you. The purpose and usage of the namespace will make more sense as you start using it. For now, move on and use the pattern as I give it to you.

We'll be talking about that namespace a ton, so the canonical pattern is to save that namespace dictionary as a variable that we can reference elsewhere. We can same this now so we can reference it elsewhere.

The namespace dictionary can have multiple namespaces declared, where the alias is the key (as a string) and the value is the URL (as a string) as seen in the file. You may use any alias you would like for that namespace, but the URL must perfectly match what appears in the file.

Provide as many alias: URL pairs as you need for your document. Most namespaces have a canonical alias to use, which you should abide by when possible.

In [63]:
```
ns = {'tei': 'http://www.tei-c.org/ns/1.0'}
```

Evaluating an extraction query to get a single result

Remember that all your previous queries all needed to end with an extraction function at the end. This was likely either `/text()` to get the text of the element out, or `@attribute` to get some attribute text out.

For example, `//a` would select all the `a` element nodes, but not yield the contents. But `//a/text()` would give you the hyperlink text, and `//a/@href` would give you all the URLs for the hyperlinks.

The reasons for this aren't always made clear by those GUI tools we were using. However, the distinction between a selection and extraction query will be striking when using this tool in Python. You **must** include an extraction statement in your query to get text content out. Otherwise you'll be selecting elements, and nothing will appear useful in the list of results.

As a start, we're going to run a query that will extract out a single result. We're going to look up the standard name of Hamlet from his character data node.

The xpath that we would want to use is `//person[@xml:id = "F-ham-ham"]/persName[@type = "standard"]/text()`, but we need to adapt this to our namespace. Look back up to our `ns` dictionary and look at what we declared the alias to be. We gave our TEI namespace schema an alias of `tei`, which means we need to provide this before each element name that we are referencing. IMPORTANT! You only need to do this for element names, not for attribute values, content, or XPath functions. Literally only for the element names, but for **every** element name. Even when you have multiple.

So now our new XPath query with correct aliases will be:

```
//tei:person[@xml:id = "F-ham-ham"]/tei:persName[@type = "standard"]/text()
```

See those `tei:person` and `tei:persName`? That's how you use that alias value. It's `alias:element`.

Let's put this together and see the results.

In [64]:
```
print(tree.xpath('//tei:person[@xml:id = "F-ham-ham"]/tei:persName[@type = "st
andard"]/text()', namespaces = ns))
```

```
['Hamlet, son of the former king and nephew to the\n
   present king']
```

Things to note:

- I am using my alias here only for the elements, and that alias name matches what I have declared in my `ns` object.
- I have `namespaces = ns` which will need to be in **each and every xpath query you run for this assignment**.
- my xpath query is just a string
- I've used double quotes in my xpath query, which means that I need to use single quotes to surround the string.
- my results are coming back as a list with one element. I know and expect there to be just a single result, but the results will always be coming back to you as a list.
- that extra text is from a the newline in the XML file itself.

Query to extract many results

Let's adapt our previous result to find all the standard names for these characters. We don't need to change much. We need to take out the `@xml:id = "F-ham-ham"` that selected only Hamlet's node, and now it will select all the person nodes.

In [65]:
```
results = tree.xpath('//tei:person/tei:persName[@type = "standard"]/text()', n
amespaces = ns)
print(results)
```

```
['First Player', 'All', 'Ambassador', 'Player Prologue', 'Player Queen', 'Ber
nardo, sentinel', 'Norwegian Captain', 'First Clown', 'Fortinbras, Prince of
', 'Francisco, a soldier', 'Gentleman, courtier', 'Gentlemen', "Father's Ghos
t, Ghost of Hamlet's\n                              Father", 'Guildenstern, cou
rtier', 'Hamlet, son of the former king and nephew to the\n
          present king', 'Horatio, friend to Hamlet', 'Claudius, King of Denm
ark', 'Laertes, son to Polonius', 'Lucianus', 'Marcellus, Officer', 'Messenge
r', 'Ophelia, daughter to Polonius', 'Osric, courtier', 'Second Clown', 'Polo
nius, Lord Chamberlain', 'Player King', 'Priest', 'Gertrude, Queen of Denmark
and mother to\n                            Hamlet', 'Rosencrantz, courtier',
'Reynaldo, servant to Polonius', 'Sailor', 'Servant', 'Voltemand, courtier']
```

Now we have a list of results to play with!

How many characters have standard names?

In [66]:
```
print(len(results))
```

```
33
```

Loop through the names and normalize the spaces.

In [67]:
```
for name in results:
    print(" ".join(name.split()))
```

```
First Player
All
Ambassador
Player Prologue
Player Queen
Bernardo, sentinel
Norwegian Captain
First Clown
Fortinbras, Prince of
Francisco, a soldier
Gentleman, courtier
Gentlemen
Father's Ghost, Ghost of Hamlet's Father
Guildenstern, courtier
Hamlet, son of the former king and nephew to the present king
Horatio, friend to Hamlet
Claudius, King of Denmark
Laertes, son to Polonius
Lucianus
Marcellus, Officer
Messenger
Ophelia, daughter to Polonius
Osric, courtier
Second Clown
Polonius, Lord Chamberlain
Player King
Priest
Gertrude, Queen of Denmark and mother to Hamlet
Rosencrantz, courtier
Reynaldo, servant to Polonius
Sailor
Servant
Voltemand, courtier
```

Profiling structures

You can't be an expert in all schemas, so sometimes you need to use some tools in python to profile the data that you are working with.

We can look inside the Hamlet person node and see that there are 4 reported variations:

```
<persName type="form">Ha.</persName>
<persName type="form">Ham.</persName>
<persName type="form">Hamlet.</persName>
<persName type="form">Hem.</persName>
```

But can we confirm that this really is the case? Alternatively, what if we were the ones writing this data file and needed to fill this in? Also, this doesn't include the counts, so we don't really know the distribution of these forms.

Let's write a query that finds all the speaker representations of Hamlet, and then runs the results through the couter tool that we've seen before.

Here's our xpath to find all of Hamlet's passages:

```
//tei:sp[@who = "#F-ham-ham"]
```

Now find all the speaker elements in there.

```
//tei:sp[@who = "#F-ham-ham"]/tei:speaker
```

Now get all that text out!

```
//tei:sp[@who = "#F-ham-ham"]/tei:speaker/text()
```

In [68]:
```
results = tree.xpath('//tei:sp[@who = "#F-ham-ham"]/tei:speaker/text()', names
paces = ns)
print(results)
print(len(results))
```

```
['Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ha.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Hamlet.', 'Ham.', 'Ham.', 'Ham.', 'H
am.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Hem.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.',
'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ha
m.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.', 'Ham.']
340
```

There's 340 of these, which would be super annoying to count by hand.

In [69]:
```
from collections import Counter

print(Counter(results))
```

```
Counter({'Ham.': 336, 'Ha.': 2, 'Hamlet.': 1, 'Hem.': 1})
```

So now we know more about this data! And in just a few lines of code.

Selecting nodes

Up to now, we've been focusing on the extraction of data. However, this tool is much more powerful than that. As we've discussed with other data structures in previous lectures, sometimes it can be really valuable to isolate the specific data granularity that you want. Once you have those chunks isolated, you can drill down into them to get out information that you want. We can do the same thing here.

The value of being able to select just a node (instead of extracting information out of it) is that you can save that object node as a variable and apply xpath queries directly onto it. Yes, we could always include that information in our original xpath if we were wanting a single value. But sometimes we want more.

However, when we can isolate a node we can run however many xpath queries we want on that node. And this is why it is powerful.

Some of the examples that we will be going through below could also be done with xpath functions, but those aren't always supported inside these packages. Also, this lesson is meant to highlight brining in data into python.

So with that said, let's explore this.

You can easily select just the nodes for your query by omitting the extraction chunk of your query.

We're curious about stage directions in speaker elements. These directions have both text that we want and attribute values. We could write this in two queries.

In [70]:
```
stagedirtype = tree.xpath('//tei:sp/tei:stage/@type', namespaces=ns)
```

In [71]:
```
stagedirtext = tree.xpath('//tei:sp/tei:stage/text()', namespaces=ns)
```

In [72]:
```
print(len(stagedirtype))
print(len(stagedirtext))
```

```
30
33
```

Hmmm, so if we did this with two queries, we can see that there are differing length results. This means that the results don't line up via positions, and there aren't ways that I can predict or know by just looking at the content. So doing this as two separate passes won't work.

And indeed, there are some stage elements that do not have type attributes. Example: `<stage rend="italic inline">within.</stage>`

Using this structure of gathering all the elements and then extracting the content allows us to navigate this kind of situation and provides protection when we might not expect that to be the case.

Let's rewrite our query such that we only get stage elements that have the type attribute (of any value). This time, we're only going to select the matching elements, and not extract anything.

To check that an element contains an attribute value, we can place that attribute reference in the logical check area, but with no operators. We can select and gather all the elements by omitting any extraction notations on the end.

In [73]:
```
stagedirswithtype = tree.xpath('//tei:sp/tei:stage[@type]', namespaces=ns)
```

In [74]:
```
stagedirswithtype
```

Out[74]:
```
[<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595199408>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595199108>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595103748>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e5951907c8>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595199148>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190248>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190dc8>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190148>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190fc8>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190208>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190448>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190508>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190e88>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e5951908c8>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190088>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185308>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185508>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190e48>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185608>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185348>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185548>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185b08>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185b48>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e5951851c8>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e5951859c8>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e5951853c8>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185908>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185c88>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e5951858c8>,
 <Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185e48>]
```

And when we print this out, we don't see text. We see that we are storing objects in memory that have a nice method of printing (because what would you print?). That's what that <> thing means around them. We have Element objects stored, but we are getting the default string representation.

While this might look like an error, it is exactly what we want!

We now have a list of objects, and we want to loop over them. Well, we don't need anything fancy for that.

In [75]:
```
for stageelem in stagedirswithtype:
    print(stageelem)
```

```
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595199408>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595199108>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595103748>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e5951907c8>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595199148>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190248>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190dc8>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190148>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190fc8>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190208>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190448>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190508>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190e88>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e5951908c8>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190088>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185308>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185508>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595190e48>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185608>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185348>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185548>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185b08>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185b48>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e5951851c8>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e5951859c8>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e5951853c8>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185908>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185c88>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e5951858c8>
<Element {http://www.tei-c.org/ns/1.0}stage at 0x2e595185e48>
```

Just because they don't have a pretty print out doesn't mean that we're doing something wrong. These objects all represent the stage elements that we selected, and can have xpath queries run directly on them.

Pretty much all your previous xpath query stuyy will work on them as normal, with one exception: there is no root to handle here. So whatever you run, it will be relevant to that element that you have selected. Operationalizing this, it means that your xpath exressions don't need to start with / or //, and effectively only need to be the piece you would add on to an existing expression if you hadn't already selected them.

Let's take an HTML example here.

If I selected all a elements within a table with `//table//a`, but what I really wanted was the `href` attribute content, then my xpath query would only be `@href` because the element is already selected (as in, it's the object that I'm already iterating over).

So now that we have our speaker elements, we want to loop over them to collect multiple pieces of information about them. In this case, we want the speaker ID, what type of direction it is, and then what the text of the direction says.

We'll build this up. The easiest is adding the `@type` extraction onto this.

In [76]:
```
for stage in stagedirswithtype:
    print(stage.xpath('@type', namespaces = ns))
```

```
['entrance']
['entrance']
['exit']
['entrance']
['exit']
['entrance']
['business']
['business']
['exit']
['entrance']
['exit']
['entrance']
['exit']
['business']
['entrance']
['entrance']
['entrance']
['exit']
['location']
['entrance']
['business']
['business']
['entrance']
['exit']
['business']
['business']
['business']
['entrance']
['business']
['business']
```

In [77]:
```
for stage in stagedirswithtype:
    print(stage.xpath('@type', namespaces = ns), stage.xpath('text()', namespa
ces = ns))
```

```
['entrance'] ['Enter the\n                                    Ghost.']
['entrance'] ['Enter Ghost againe.']
['exit'] ['Exit Ghost.']
['entrance'] ['Enter Voltemand and\n                                  Corneliu
s.']
['exit'] ['Exit Voltemand and\n                                Cornelius.']
['entrance'] ['Enter Polonius.']
['business'] ['within.']
['business'] ['The Letter.']
['exit'] ['Exit King &\n                                Queen.']
['entrance'] ['Enter foure or fiue\n                                Player
s.']
['exit'] ['Exit Players.']
['entrance'] ['Enter Polonius, Rosincrance,\n
and Guildensterne.']
['exit'] ['Exit Polonius.']
['business'] ['Sleepes']
['entrance'] ['Enter Lucianus.']
['entrance'] ['Enter one with a\n                                Recorder.']
['entrance'] ['Enter Ros. &\n                              Guild.']
['exit'] ['Exit Gent.']
['location'] ['within.']
['entrance'] ['Enter ', '.']
['business'] ['A noise\n                                within.']
['business'] ['Reads the Letter.']
['entrance'] ['Enter a Messenger.']
['exit'] ['Exit Messenger']
['business'] ['Sings.']
['business'] ['sings.']
['business'] ['sings.']
['entrance'] ['Enter King, Queen, Laertes,\n                                  a
nd a Coffin, ', 'with Lords attendant.']
['business'] ['Leaps in the\n                              graue.']
['business'] ['March afarre off,\n                                and shout w
ithin.']
```

We've got our two results now, but getting the person's ID might be trickier. Well, not really, but it might look weird. Remember that we have selected the stage elements, but the @who attribute is within the parent sp element that we didn't select.

That's the wonderful thing about this, these are element objects, and they know all about their parent elements. We can use .. as the beginning of our query, just as if we were using it inside a longer xpath expression.

In [78]:
```
for stage in stagedirswithtype:
    print(stage.xpath('../@who', namespaces = ns))
    print(stage.xpath('@type', namespaces = ns), stage.xpath('text()', namespa
ces = ns))
```

```
['#F-ham-mar']
['entrance'] ['Enter the\n                                    Ghost.']
['#F-ham-hor']
['entrance'] ['Enter Ghost againe.']
['#F-ham-mar']
['exit'] ['Exit Ghost.']
['#F-ham-cla']
['entrance'] ['Enter Voltemand and\n                                Corneliu
s.']
['#F-ham-cla']
['exit'] ['Exit Voltemand and\n                                Cornelius.']
['#F-ham-lae']
['entrance'] ['Enter Polonius.']
['#F-ham-hor #F-ham-mar']
['business'] ['within.']
['#F-ham-pol']
['business'] ['The Letter.']
['#F-ham-pol']
['exit'] ['Exit King &\n                                Queen.']
['#F-ham-ham']
['entrance'] ['Enter foure or fiue\n                                Player
s.']
['#F-ham-ham']
['exit'] ['Exit Players.']
['#F-ham-ham']
['entrance'] ['Enter Polonius, Rosincrance,\n
and Guildensterne.']
['#F-ham-ham']
['exit'] ['Exit Polonius.']
['#F-ham-ger']
['business'] ['Sleepes']
['#F-ham-ham']
['entrance'] ['Enter Lucianus.']
['#F-ham-ham']
['entrance'] ['Enter one with a\n                                Recorder.']
['#F-ham-cla']
['entrance'] ['Enter Ros. &\n                                Guild.']
['#F-ham-cla']
['exit'] ['Exit Gent.']
['#F-ham-gmn']
['location'] ['within.']
['#F-ham-cla']
['entrance'] ['Enter ', '.']
['#F-ham-cla']
['business'] ['A noise\n                                within.']
['#F-ham-sai']
['business'] ['Reads the Letter.']
['#F-ham-cla']
['entrance'] ['Enter a Messenger.']
['#F-ham-cla']
['exit'] ['Exit Messenger']
['#F-ham-clo.1']
['business'] ['Sings.']
['#F-ham-clo.1']
['business'] ['sings.']
['#F-ham-clo.1']
['business'] ['sings.']
```

```
['#F-ham-ham']
['entrance'] ['Enter King, Queen, Laertes,\n                                    a
nd a Coffin, ', 'with Lords attendant.']
['#F-ham-lae']
['business'] ['Leaps in the\n                                graue.']
['#F-ham-ham']
['business'] ['March afarre off,\n                                    and shout w
ithin.']
```

making the results pretty

Now we've got a host of results, but the print is really ugly. We can collect everything that we want into a single list for nice printing. Remember that everything is inside of a list, so we need to extract out the results.

In [79]:
```
for stage in stagedirswithtype:
    results = []
    who = stage.xpath('../@who', namespaces = ns)
    dirtype = stage.xpath('@type', namespaces = ns)
    dirtext = stage.xpath('text()', namespaces = ns)
    results.append(who[0])
    results.append(dirtype[0])
    results.append(" ".join(dirtext[0].split()))
    print(results)
```

```
['#F-ham-mar', 'entrance', 'Enter the Ghost.']
['#F-ham-hor', 'entrance', 'Enter Ghost againe.']
['#F-ham-mar', 'exit', 'Exit Ghost.']
['#F-ham-cla', 'entrance', 'Enter Voltemand and Cornelius.']
['#F-ham-cla', 'exit', 'Exit Voltemand and Cornelius.']
['#F-ham-lae', 'entrance', 'Enter Polonius.']
['#F-ham-hor #F-ham-mar', 'business', 'within.']
['#F-ham-pol', 'business', 'The Letter.']
['#F-ham-pol', 'exit', 'Exit King & Queen.']
['#F-ham-ham', 'entrance', 'Enter foure or fiue Players.']
['#F-ham-ham', 'exit', 'Exit Players.']
['#F-ham-ham', 'entrance', 'Enter Polonius, Rosincrance, and Guildensterne.']
['#F-ham-ham', 'exit', 'Exit Polonius.']
['#F-ham-ger', 'business', 'Sleepes']
['#F-ham-ham', 'entrance', 'Enter Lucianus.']
['#F-ham-ham', 'entrance', 'Enter one with a Recorder.']
['#F-ham-cla', 'entrance', 'Enter Ros. & Guild.']
['#F-ham-cla', 'exit', 'Exit Gent.']
['#F-ham-gmn', 'location', 'within.']
['#F-ham-cla', 'entrance', 'Enter']
['#F-ham-cla', 'business', 'A noise within.']
['#F-ham-sai', 'business', 'Reads the Letter.']
['#F-ham-cla', 'entrance', 'Enter a Messenger.']
['#F-ham-cla', 'exit', 'Exit Messenger']
['#F-ham-clo.1', 'business', 'Sings.']
['#F-ham-clo.1', 'business', 'sings.']
['#F-ham-clo.1', 'business', 'sings.']
['#F-ham-ham', 'entrance', 'Enter King, Queen, Laertes, and a Coffin,']
['#F-ham-lae', 'business', 'Leaps in the graue.']
['#F-ham-ham', 'business', 'March afarre off, and shout within.']
```

This has all worked nicely because everything had at least one value, so we can hard code that [0] pretty safely. But this will not always we the case. We can look up and see that one of the items has two speakers, and the XML authors chose to handle that by listing them both in the @who as "#F-ham-hor #F-ham-mar". So there's still only 1 result in the sense that there's only one thing coming back to us with our query (because you can't repeat attributes in a single element). So they added both results to a single string and made that the single attribute's value.

Handling empty results

This is fine for now. We would need to accommodate our data model for this if we were really doing research here.

However, we could take a look back to our original 33 results. Not all of these had a type, right? So what happens if our query has no results?

You get an empty list.

In [80]:
```
print(tree.xpath('tei:frogs/text()', namespaces=ns))
```

```
[]
```

How you want to handle this really depends on what you are after and your own business rules for the data that you are after. Usually you can put in a little decision structure to check the length (or content if you need). This is where your knowledge of the data will be very important.

Let's use a function for checking that out results have only one item. Otherwise, it will raise an error if it gets more, and fill in a missing value if there is nothing.

Now, this function can't tell you WHY there is nothing, so again, know your data and test your code.

Let's explore this function that I've prepared for you. This actually has nothing to do with xpath at all, as the results that we get back will be purely as a list.

In [81]:
```
def checkFor1Result(xpathresult, missing_value):
    if len(xpathresult) > 1: # raise error if there are more than 1
        howmany = len(xpathresult)
        raise ValueError("Your list had " + str(howmany) + " items instead of
 1. Shutting down the program,"
                         + "But here's your failed result: " + str(xpathresult
))
    elif len(xpathresult) == 1:
        result = xpathresult[0] # send the single result back when there's jus
t 1
    else: # send the missing value back when there aren't any results
        result = missing_value
    return result
```

In [82]:
```
# raises an error and halts the program if there are more than 1
print(checkFor1Result(['too', 'many'], 'MissingResult'))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-82-db2a8f58fa3a> in <module>()
      1 # raises an error and halts the program if there are more than 1
----> 2 print(checkFor1Result(['too', 'many'], 'MissingResult'))

<ipython-input-81-68c835c76142> in checkFor1Result(xpathresult, missing_valu
e)
      3         howmany = len(xpathresult)
      4         raise ValueError("Your list had " + str(howmany) + " items in
stead of 1. Shutting down the program,"
----> 5                          + "But here's your failed result: " + str(xp
athresult))
      6     elif len(xpathresult) == 1:
      7         result = xpathresult[0] # send the single result back when th
ere's just 1

ValueError: Your list had 2 items instead of 1. Shutting down the program,But
 here's your failed result: ['too', 'many']
```

In [83]:
```
# passes the value through if there are exactly 1
print(checkFor1Result(['yup'], 'MissingResult'))
```

```
yup
```

In [84]:
```
# passes the missing result through if there's nothing.
print(checkFor1Result([], 'MissingResult'))
```

```
MissingResult
```

This function guarentees that you will have 1 and only 1 result coming back to you.

Let's see this in action on all 33 stage elements.

In [85]:
```
allstages = tree.xpath('//tei:sp/tei:stage', namespaces=ns)
```

In [86]:
```
for stage in allstages:
    results = []
    who = stage.xpath('../@who', namespaces = ns)
    dirtype = stage.xpath('@type', namespaces = ns)
    dirtext = stage.xpath('.//text()', namespaces = ns)
    results.append(checkFor1Result(who, 'MissingWho'))
    results.append(checkFor1Result(dirtype, 'MissingType'))
    results.append(" ".join(checkFor1Result(dirtext, 'MissingText').split()))
    print(results)
```

```
['#F-ham-mar', 'entrance', 'Enter the Ghost.']
['#F-ham-hor', 'entrance', 'Enter Ghost againe.']
['#F-ham-mar', 'exit', 'Exit Ghost.']
['#F-ham-cla', 'entrance', 'Enter Voltemand and Cornelius.']
['#F-ham-cla', 'exit', 'Exit Voltemand and Cornelius.']
['#F-ham-lae', 'entrance', 'Enter Polonius.']
['#F-ham-hor #F-ham-mar', 'business', 'within.']
['#F-ham-pol', 'business', 'The Letter.']
['#F-ham-pol', 'exit', 'Exit King & Queen.']
['#F-ham-ham', 'entrance', 'Enter foure or fiue Players.']
['#F-ham-ham', 'exit', 'Exit Players.']
['#F-ham-ham', 'entrance', 'Enter Polonius, Rosincrance, and Guildensterne.']
['#F-ham-ham', 'exit', 'Exit Polonius.']
['#F-ham-ger', 'business', 'Sleepes']
['#F-ham-ham', 'entrance', 'Enter Lucianus.']
['#F-ham-ham', 'entrance', 'Enter one with a Recorder.']
['#F-ham-ham', 'MissingType', 'within.']
['#F-ham-cla', 'entrance', 'Enter Ros. & Guild.']
['#F-ham-cla', 'exit', 'Exit Gent.']
['#F-ham-gmn', 'location', 'within.']

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-86-a4442d490155> in <module>()
      6     results.append(checkFor1Result(who, 'MissingWho'))
      7     results.append(checkFor1Result(dirtype, 'MissingType'))
----> 8     results.append(" ".join(checkFor1Result(dirtext, 'MissingText').s
plit()))
      9     print(results)

<ipython-input-81-68c835c76142> in checkFor1Result(xpathresult, missing_valu
e)
      3         howmany = len(xpathresult)
      4         raise ValueError("Your list had " + str(howmany) + " items in
stead of 1. Shutting down the program,"
----> 5                          + "But here's your failed result: " + str(xp
athresult))
      6     elif len(xpathresult) == 1:
      7         result = xpathresult[0] # send the single result back when th
ere's just 1

ValueError: Your list had 7 items instead of 1. Shutting down the program,But
 here's your failed result: ['Enter ', '\n
', 'Rosincrane', '\n                                    ', 'Rosincrance',
 '\n                               ', '.']
```

Oh fun! We can see that we have an error that was hidden from us by our previous method. This is becuase there's formatted text inside that element. We can handle the results together before passing it to the function. But remember that it must be a list!

In [87]:
```
for stage in allstages:
    results = []
    who = stage.xpath('../@who', namespaces = ns)
    dirtype = stage.xpath('@type', namespaces = ns)
    dirtext = stage.xpath('.//text()', namespaces = ns)
    results.append(checkFor1Result(who, 'MissingWho'))
    results.append(checkFor1Result(dirtype, 'MissingType'))
    results.append(" ".join(checkFor1Result([" ".join(dirtext)], 'MissingText'
).split()))
    print(results)
```

```
['#F-ham-mar', 'entrance', 'Enter the Ghost.']
['#F-ham-hor', 'entrance', 'Enter Ghost againe.']
['#F-ham-mar', 'exit', 'Exit Ghost.']
['#F-ham-cla', 'entrance', 'Enter Voltemand and Cornelius.']
['#F-ham-cla', 'exit', 'Exit Voltemand and Cornelius.']
['#F-ham-lae', 'entrance', 'Enter Polonius.']
['#F-ham-hor #F-ham-mar', 'business', 'within.']
['#F-ham-pol', 'business', 'The Letter.']
['#F-ham-pol', 'exit', 'Exit King & Queen.']
['#F-ham-ham', 'entrance', 'Enter foure or fiue Players.']
['#F-ham-ham', 'exit', 'Exit Players.']
['#F-ham-ham', 'entrance', 'Enter Polonius, Rosincrance, and Guildensterne.']
['#F-ham-ham', 'exit', 'Exit Polonius.']
['#F-ham-ger', 'business', 'Sleepes']
['#F-ham-ham', 'entrance', 'Enter Lucianus.']
['#F-ham-ham', 'entrance', 'Enter one with a Recorder.']
['#F-ham-ham', 'MissingType', 'within.']
['#F-ham-cla', 'entrance', 'Enter Ros. & Guild.']
['#F-ham-cla', 'exit', 'Exit Gent.']
['#F-ham-gmn', 'location', 'within.']
['#F-ham-cla', 'entrance', 'Enter Rosincrane Rosincrance .']
['#F-ham-cla', 'business', 'A noise within.']
['#F-ham-sai', 'business', 'Reads the Letter.']
['#F-ham-cla', 'entrance', 'Enter a Messenger.']
['#F-ham-cla', 'exit', 'Exit Messenger']
['#F-ham-clo.1', 'business', 'Sings.']
['#F-ham-clo.1', 'business', 'sings.']
['#F-ham-clo.1', 'business', 'sings.']
['#F-ham-ham', 'entrance', 'Enter King, Queen, Laertes, and a Coffin, with Lo
rds attendant.']
['#F-ham-lae', 'business', 'Leaps in the graue.']
['#F-ham-ham', 'business', 'March afarre off, and shout within.']
```

This in coming from this item on line 5934:

```
<stage rend="italic center" type="entrance">Enter <choice>
                                    <orig>Rosincrane</orig>
                                    <corr>Rosincrance</corr>
                                </choice>.</stage>
```

If we were really wanting to do something with this, we would want to deal with that. But we are not, so we can let it go.

Exporting results

Now that we have some nice tabular results, we can export this out to a file!

We can use the CSV module to write out a nice CSV. This will automatically sanitize our results for us. You'll need to have two things:

1. a list with the string values of the headers that you want (one string per header).
   - `['speakerID', 'directionType', 'directionText']`
2. a list of lists, where each list contains a single row of data. Each element will become a different column.
   - we are already making our row lists, we just need to collect them.

In [88]:
```
allresults = []

for stage in allstages:
    results = []
    who = stage.xpath('../@who', namespaces = ns)
    dirtype = stage.xpath('@type', namespaces = ns)
    dirtext = stage.xpath('.//text()', namespaces = ns)
    results.append(checkFor1Result(who, 'MissingWho'))
    results.append(checkFor1Result(dirtype, 'MissingType'))
    results.append(" ".join(checkFor1Result([" ".join(dirtext)], 'MissingText'
).split()))
    allresults.append(results)
```

In [89]:
```
headers = ['speakerID', 'directionType', 'directionText']
```

In [90]:
```
import csv

outfile = open('stagedirections.csv', 'w')
csvout = csv.writer(outfile)
csvout.writerow(headers)
csvout.writerows(allresults)
```

And we're done!