```
== Learning XPath through Examples ==

= Basics =

General idea of paths is familiar from unix
- ls course-*/assignments/a1/solution/*.py


Simplest example
- fn:doc("quiz.xml")/quiz/questions/mc-question/question
        - just a straight-up path
        - notice that you get a list of results
        - comma-separated
        - they are in the order in which they appear in the doc
        - it is case sensitive: break it and see


Going up the tree
- modify it to go one node less deep and see result
- Aside: what if you create a path that doesn't exist
        - eg, take out a step
        - get the empty list
- keep shortening the path until the path is empty:
        - you get the document node


fn:doc("quiz.xml")/quiz vs fn:doc("quiz.xml")
- fn:doc("quiz.xml")/quiz
        - a sequence of one item
        - that item is the node for the element that is the document's root
          (quiz), and contains the whole document within it
- fn:doc("quiz.xml")
        - a sequence of one item
        - that item is the document node for the whole file
- the difference is clear if you go back to the picture of the tree
  that the xml represents [see the "XML & DTD" lecture slides]


Using the text() function
- if you add text() at the end, you get the body of the element,
  not the element.
- eg where the element contains #PCDATA
        - fn:doc("quiz.xml")/quiz/questions/mc-question/question/text()
        - get text of the body of the question elements
- eg where it doesn't
        - take out "questions"
        - the text part of an mc-question element is just some whitespace
        - notice inconsistencies in the whitespace
        - quiz-a.xml has the whitespace manipulated in interesting ways.
        - try running the same query on quiz-a.xml to see the effects.
          fn:doc("quiz-a.xml")/quiz/questions/mc-question/question/text()


Go to an attribute
- @att-name  (and you do need a slash before it)
- fn:doc("quiz.xml")/quiz/questions/mc-question/@solution
- get primitive-type result, not node-type
   attribute solution {"1"}, attribute solution {"4"}, attribute solution {"3"}
- another example:
  fn:doc("quiz.xml")/quiz/class-responses/student/@sid
        attribute sid {"s998801234"},
        attribute sid {"s001078452"},
        attribute sid {"s997733991"},
        attribute sid {"s555555555"}
```

```
- can't go further on the path because all attribute nodes are leaves
  in the document tree


= More features =

Start at any blah-node
- begin path with //blah
- meaning: begin by finding any blah node anywhere at root or lower
- Saves a lot of typing!
  Also handy if you can't remember or don't want to look up a long path
  to a node.
- fn:doc("quiz.xml")//question
        Notice that a question node can occur in two different contexts.
- fn:doc("quiz.xml")//@sid
        Here it's an attribute we're finding
- fn:doc("quiz.xml")//@qid
        Why do we see the same qid values repeated so many times?
- can actually put // anywhere in the path expression
        fleep...floop//blah
  Start out following the path expression to floop, then from there find
  a blah-node ANYWHERE underneath
- Can continue the path expression afterward
- fn:doc("quiz.xml")//mc-question/@qid
        qid values but only in an mc-question
- fn:doc("quiz.xml")//questions//@qid
        qid values but only under some question node
- fn:doc("quiz.xml")/quiz/class-responses//@qid
        qid values but only somewhere under a class-responses node


Wildcard
- fn:doc("quiz.xml")/*/@qid
        - This yields nothing!
- The reason is that * represents any one tag.
- fn:doc("quiz.xml")/*/*/*/@qid
        - This will only match a qid attribute that is at level 4.
- fn:doc("quiz.xml")/*/*/*/*/@qid
        - This will only match a qid attribute that is at level 5.



Getting heterogeneous results
- Eg:
        fn:doc("quiz.xml")//questions/*/*
- If you have an element e with different kinds of subelement,
  path expression
        .../e/*
  will give you results that include those two different kinds of subelement.
- This is one way to get different kinds of items in the resulting output.
- here we get question elements and option elements
  And notice that they come from different kinds of contexts
  (mc-question and tf-question).


Indexing into a node set
- Eg:
        fn:doc("quiz.xml")//class-responses/student
        fn:doc("quiz.xml")//class-responses/student[1]
- note that it counts from 1, not 0.
- and if the index isn't valid?
        fn:doc("quiz.xml")//class-responses/student[22]


= Selection conditions =
Selection conditions
- prunes the paths that have been found
- Append this to the end of the path expression:
        [ expression ]


Examples with attributes
- within the [ ]-ed expression, use
        @att-name to refer to the value of an attribute of the current element
- Go down to mc-question and only keep those whose qid attribute is >= "Q555"
        fn:doc("quiz.xml")//mc-question[@qid >= "Q555"]
- One with a compound condition
        fn:doc("quiz.xml")//response[@qid="Q888" and @answer !="4"]


Examples with elements
- within the [ ]-ed expression, use
        subelementName to refer to a subelement of the current element
        "." to refer to the entirety of the current element
- Go down to a race element, and
  only keep those with a result element with value <= 3.50
        fn:doc("races.xml")/races/race[result <= 3.50]
- It's existentially quantified: One or more such race elements will do
- What you get back is race elements that satisfy the condition
- Go down to result elements and
  only keep those that themselves (.) have value <= 3.50
        fn:doc("races.xml")/races/race/result[. <= 3.50]
- What you get back is result elements that satisfy the condition


Can compare two attributes/elements -- not just compare to constants
- Here we compare an attribute and an element
        fn:doc("races.xml")/races/race[@name = sponsor]
- in this dtd, can even compare the same element to itself
        fn:doc("races.xml")/races/race[result > result]
- It's existentially quantified: One or pairs of such result elements will do
- Notice it doesn't report DawnAtTheDon's,
  because there are no two results where one is > the other.


Can continue the path expression beyond the selection
- Example: Find races with names in a certain range,
  then pull out any result below
        fn:doc("races.xml")/races/*[@name > "K"]/result
- Notice that we get every result element that's under one of the elements
  found by
        fn:doc("races.xml")/races/*[@name > "K"]


= Cool functions =

- See http://www.w3.org/TR/xpath-functions
- average etc:
        fn:doc("races.xml")/races/race/avg(result)
        fn:doc("races.xml")/races/race/max(result)
        fn:doc("races.xml")/races/race/count(result)
  Why do we get multiple answers??
        For each *race* element, we are asking for a count of
        its result sub-elements.
        (There is only one race element.)
- A slightly different use of count:
        fn:doc("races.xml")/races/count(race)
  Why do we now get one answer??
        For each *races* element, we are asking for a count of
        its race sub-selements.
        (There are several races elements.)
- Using these functions in a condition
    fn:doc("races.xml")/races/race[avg(result)>4.50]
    fn:doc("races.xml")/races/race[avg(result)>4.50 or min(result)>2.50]
- Then can continue the path expression
    fn:doc("races.xml")/races/race[avg(result)>4.50 or min(result)>2.50]/result
- To get the overall average:
```