Gradiance Online Accelerated Learning

Submission number: 151329
Submission certificate: BH521004
Submission time: 2014-02-10 01:53:53 PST (GMT - 8:00)

Number of questions: 9
Positive points per question: 3.0
Negative points per question: 1.0
Your score: 27

1. Study the following XML document and select, from the list below, the XPath expression that evaluates to 'true'.

```
<a>
  <b>
    <c/>
    <d/>
  </b>
  <e>
    <f>
      <g/>
    </f>
    <h>
      <i>
        <j/>
        <k/>
      </i>
      <l>
        <m/>
      </l>
    </h>
  </e>
  <n>
    <o/>
    <p>
      <q/>
    </p>
  </n>
</a>
```

Note: The "count()" method returns the number of nodes in a node-set. Also, the '|' operator computes the union of two node-sets.

a) `count(//h/preceding::node() | //h/preceding-sibling::node()) = 6`

b) `count(//h/ancestor-or-self::node() | //h/descendant-or-self::node()) = 8`

c) `count(//h/preceding::node() | //h/preceding-sibling::node()) = 5`

d) `count(//h/following::node() | //h/following-sibling::node()) = 9`

Answer submitted: **c)**

You have answered the question correctly.

Question Explanation:

The XPath recommendation specifies the following axes for identifying a set of nodes relative to a context node in the XML document:

- The **self** axis contains just the context node itself.

- the **descendant** axis contains the descendants of the context node.
- The **parent** axis contains the parent of the context node, if there is one.
- The **ancestor** axis contains the ancestors of the context node. The ancestors of the context node consist of the parent of the context node and the parent's parent and so on. Note that the ancestor axis will always include the root node, unless the context node is the root node.
- The **descendant-or-self** axis contains the context node and the descendants of the context node.
- The **ancestor-or-self** axis contains the context node and the ancestors of the context node. Note that the ancestor axis will always include the root node.
- The **following** axis contains all nodes in the document that occur after the context node in document order, i.e., their start tag occurs after the end tag of the context node. Thus, this axis excludes any descendants.
- The **preceding** axis contains all nodes in the document that occur before the context node in document order, i.e., their end tag occurs before the start tag of the context node. Thus, this axis excludes any ancestors.
- The **following-sibling** axis contains all the following siblings of the context node.
- The **preceding-sibling** axis contains all the preceding siblings of the context node.
- The **attribute** axis contains the attributes of the context node. This axis is empty unless the context node is an element.
- The **namespace** axis contains the namespace nodes of the context node. This axis is empty unless the context node is an element.

Here are two expressions that evaluate to 'true' for the given document:

```
count(//h/ancestor-or-self::node() | //h/descendant-or-
self::node()) = 9
```

In this case, the XPath expression compares the number 9 with the number of nodes in the union of two node-sets. The context node for both node-sets in the union is the element "h". The first node-set contains the elements: h, e, a, and the document root. The second node-set has the following elements: h, i, j, k, l, m. Thus, the union contains 9 elements in all: h, e, a, i, j, k, l, m, and the document root.

```
count(//h/preceding::node() | //h/preceding-sibling::node()) = 5
```

In this case, the XPath expression compares the number 5 with the number of nodes in the union of two node-sets. The context node for both node-sets in the union is the element "h". The first node-set contains the elements: f, g, b, d, c. The second node-set has just one element: f. Thus, the union contains 5 elements in all: f, g, b, d, c.

For more details on XPath expressions, please refer to the W3C XML Path Language recommendation.

---

2. Here is a document:

```
      <EMPS>
          <EMP name = "Kermit">
              <ADDR>123 Sesame St.</ADDR>
p1            <PHONE type = "cell">555-1212</PHONE>
p2            <PHONE type = "wired">555-1234</PHONE>
          </EMP>
          <EMP name = "BettyLou">
              <ADDR>A Farm Somewhere</ADDR>
p3            <PHONE>555-3456</PHONE>
          </EMP>
      </EMPS>
```

Note that p1, p2, and p3 refer to the three PHONE elements and are not part of the document.

Evaluate the following XPath expressions:

1. //PHONE[@type = "cell"]
2. /EMPS/EMP/PHONE/@type

3. /EMPS/EMP/PHONE[@type]

Then, indicate the correct value for one of these expressions from the list below. We assume that sequences are represented by parenthesized, comma-separated lists, e.g.: (a, b, c). We use p1, p2, and p3 to represent the entire corresponding object. For example, p1 is shorthand for `<PHONE type = "cell">555-1212</PHONE>`.

a) //PHONE[@type = "cell"] = 555-1212

b) //PHONE[@type = "cell"] = <TYPE>cell</TYPE>

c) /EMPS/EMP/PHONE[@type] = ("cell", "wired")

d) /EMPS/EMP/PHONE/@type = ("cell", "wired")

Answer submitted: **d)**

You have answered the question correctly.

Question Explanation:

//PHONE[@type = "cell"] finds all phone elements with type = "cell". Note that // stands for any path from the root. Thus, the answer is (p1).

/EMPS/EMP/PHONE/@type finds the values of the name attribute of all PHONE elements. Thus, the value is ("cell", "wired").

/EMPS/EMP/PHONE[@type] finds all the PHONE elements that have a type --- any type. Thus, the value of this path expression is (p1, p2).

---

3. The following is part of an XML document:

```
<BAR><ADDR>101 Maple St.</ADDR>
    <PHONE>555-1212</PHONE>
    <PHONE>555-4567</PHONE>
</BAR>
```

Which of the following element declarations could be the declaration for BAR in a DTD?

a) <!ELEMENT BAR (ADDR* PHONE+)>

b) <!ELEMENT BAR (ADDR+ PHONE+ MANAGER)>

c) <!ELEMENT BAR (PHONE ADDR PHONE)>

d) <!ELEMENT BAR (ADDR NAME+ PHONE*)>

Answer submitted: **a)**

You have answered the question correctly.

Question Explanation:

Evidently there is one address subobject and two phone subobjects, in that order. Thus, the list of components for BAR must include ADDR, ADDR*, ADDR+, or ADDR? followed by PHONE* or PHONE+. Interspersed with these may be any tags that are not required to appear. That is, anything with a ? or *. Thus, we might also have components like NAME* or MANAGER?, at any point in the list.

---

4. Write a well-formed XML document that satisfies the following conditions:

- it is encoded in UTF-8 encoding

- it has a root element of the name "tasklist"
- the root element has 3 child elements whose name is "task"
- each of the "task" elements has an attribute named "name"
- assign the values "eat", "drink" and "play" to the "name" attributes for the 3 tasks

Select, from the choices below, the document that meets the above requirements.

a)
```
<?xml version="1.0" encoding="UTF-8"?>
<tasklist>
  <task name="eat">
  <task name="drink">
  <task name="play">
<tasklist>
```

b)
```
<?xml version='1.0' encoding='UTF-8'?>
<tasklist>
  <task name='eat'>
  <task name='drink'>
  <task name='play'>
<tasklist>
```

c)
```
<?xml version='1.0' encoding='UTF-8'?>
<tasklist>
  <task name='eat'/>
  <task name='drink'/>
  <task name='play'/>
</tasklist>
```

d)
```
<?xml version="1.0" encoding="UTF-16"?>
<tasklist>
  <task name="eat">
  <task name="drink">
  <task name="play">
</tasklist>
```

Answer submitted: **c)**

You have answered the question correctly.

Question Explanation:

A well-formed XML document must follow these rules:

- There must be exactly one top level element.
- All open tags must be closed.
- All tags are properly nested i.e. there are no overlapping tags.
- Attribute values must be enclosed in single or double quotes.
- If the XML document has an xml declaration, it must be the first construct in the document.
- If the xml declaration has a "version" attribute, its value must be "1.0".
- The optional "encoding" attribute of the xml declaration can be used to specify the character encoding used in the document.

Note that this list of rules is not exhaustive. Please refer to the W3C XML Recommendation for more information.

---

5. Study the following XML document and select, from the list below, the XPath expression that produces the result "2".

```
<?xml version="1.0" encoding="utf-8"?>
<tasklist>
  <task id="1" name="eat"/>
  <task id="2" name="drink" repeat="3"/>
  <task id="3" name="play"/>
</tasklist>
```

a) `/tasklist/task(2)/@id`

b) `/tasklist/task(@repeat)/@id`

c) `/tasklist/task[name='drink']/id`

d) `/tasklist/task[@name='drink']/@id`

Answer submitted: **d)**

You have answered the question correctly.

Question Explanation:

The XPath recommendation specifies the following rules for location paths:

- The XPath recommendation models an XML document as a tree of nodes with the root node referred to as "/".
- Child elements are referred to using their names preceded by "child::", e.g. "tasklist/child::task".
- The "child::" prefix can be dropped, e.g. "tasklist/task".
- Attributes are referred to using their names preceded by "attribute::", e.g. "task/attribute::id".
- The "attribute::" prefix can be abbreviated to "@", e.g. "task/@id".
- In case more than one node matches the XPath expression, it evaluates to a node-set containing all the matched nodes.
- Node-sets can be filtered using predicates, e.g. "/tasklist/task[@name='eat']". The result of applying the filter is a node-set that contains the nodes for which the predicate evaluates to true.
- Node-sets can also be indexed to select specific nodes within them, e.g. "tasklist/task[1]". The indexes are 1-based i.e. the first item in the set is at index 1.

For more details on XPath expressions, please refer to the W3C XML Path Language recommendation.

---

6. A relation R(A,B) can be represented by an XML document `r.xml`, described by the following DTD:

```
<!ELEMENT R (T*)>
<!ELEMENT T (A B)>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
```

Suppose we want to produce the sequence of <B> elements that appear in tuples with A=10. That is, we want to do the XQuery equivalent of:

```
SELECT B FROM R
WHERE A = 10;
```

Find, in the list below, the XQuery query that produces the desired result. For example, if *R* consists of the following tuples: (10,20), (5,30), (10,40), and (10,50), your query should produce, in some order, the elements <B>20</B> <B>40</B> <B>50</B>.

a)
```
for $t in doc("r.xml")/R/T[A=10]
return <B>{$t/B}</B>
```

b)
```
for $t in doc("r.xml")/R/T
where $t/A = 10
return <B>{$t/data(B)}</B>
```

c)
```
let $t := doc("r.xml")/R/T
where $t/A = 10
return <B>$t/data(B)</B>
```

d)
```
return <B>{doc("r.xml")/R/T[A=10]/data(B)}</B>
```

Answer submitted: **b)**

You have answered the question correctly.

Question Explanation:

Four possible ways to write this query are:

```
for $t in doc("r.xml")/R/T
```

```
where $t/A = 10
return $t/B

let $t := doc("r.xml")/R/T[A=10]
return $t/B

for $t in doc("r.xml")/R/T
where $t/A = 10
return <B>{$t/data(B)}</B>

return doc("r.xml")/R/T[A=10]/B
```

---

7. Study the following XML document and select, from the list below, the XPath expression that produces the result "2".

```
<?xml version="1.0" encoding="utf-8"?>
<tasklist>
  <task name="eat" priority="50" done="false"/>
  <task name="drink" priority="100" done="true" cost="10"/>
  <task name="play" priority="75" done="false"/>
  <task name="sleep" priority="25" done="true" cost="10"/>
  <task name="think" priority="5" done="false"/>
</tasklist>
```

Note: The "count()" method returns the number of nodes in a node-set.

a) `count(/tasklist/task[not(@done) = 'false'])`

b) `count(/tasklist/task[@priority > 75 or @cost])`

c) `count(/tasklist/task[@done = 'false' &amp; @priority < 75])`

d) `count(/tasklist/task[!(@done = 'false')])`

Answer submitted: **b)**

You have answered the question correctly.

Question Explanation:

The XPath recommendation specifies the following rules for location paths:

- In case more than one node matches the XPath expression, it evaluates to a node-set containing all the matched nodes.
- Node-sets can be filtered using predicates, e.g. "/tasklist/task[@name='eat']". The result of applying the filter is a node-set that contains the nodes for which the predicate evaluates to true.
- Predicates may include equality expressions using the '=' or '!=' operators e.g. "task[@done = 'true']".
- Predicates may include relational expressions using the '<', '<=', '>' or '>=' operators e.g. "task[@priority > 50]" or "task[@priority < 50]".
- Note that if you need to use an XPath expression including the '<' sign as the value of an attribute or an element in an XML document, the '<' sign must be encoded. The '>' sign in the '>' or '>=' operator may or may not be encoded.
- Predicates may include logical expressions using the 'or' or 'and' operators or the "not()" function e.g. "task[@priority > 50 and @name = 'eat']" or "task[not(@priority > 50) or @name = 'eat']".
- The default type of all attribute values and element content is "string".
- During the evaluation of predicates involving values of different types, implicit coercion is performed as necessary. The order of implicit coercion is: string -> number -> boolean i.e. strings may be coerced into numbers or boolean values and numbers may be coerced into boolean values.

Here are two expressions that produce the result "2":

```
count(/tasklist/task[@done = 'false' and @priority > 25])
```

In this case, there are 3 "task" elements with their "done" attributes set to 'false' but onlt 2 of those have their "priority" attributes set to values greater than 25. Thus, those two nodes match the predicate and the expression evaluates to "2".

```
count(/tasklist/task[@priority > 75 or @cost])
```

In this case, there are 2 "task" elements that have the "cost" attributes and one of those has its "priority" attribute set to a value greater than 75. No other "task" elements have their "priority" attribute set to a value greater than 75. Thus, those two nodes match the predicate and the expression evaluates to "2".

For more details on XPath expressions, please refer to the W3C XML Path Language recommendation.

---

8. Here is an XML prolog with DTD:

```
<?xml version="1.0"?>
<!DOCTYPE A [
    <!ELEMENT A (B+, C)>
    <!ELEMENT B (#PCDATA)>
    <!ELEMENT C (B?, D)>
    <!ELEMENT D (#PCDATA)>
]>
```

Which of the following documents matches this DTD? Note: we show only the tags (and we show them without the triangular brackets). We do not show the text that would be placed between certain pairs of tags.

a) A B /B B /B C D /D B /B /C /A

b) A B /B C B /B D /D /C /A

c) A B /B B /B C /C /A

d) A C B /B D /D /C /A

Answer submitted: **b)**

You have answered the question correctly.

Question Explanation:

An A object has within it one or more B subobjects, and then a C object. Within the C object is zero or one B followed by exactly one D. In terms of regular expressions, the tag sequences we can see are

A (B /B)(B /B)* C (D /D | B /B D /D) /C /A.

Some text may appear between each B-/B pair and each D-/D pair, but text may not appear elsewhere.

---

9. An XML document contains the following portion:

```
<EMP name = "Kermit">
    <ADDR>123 Sesame St.</ADDR>
    <PHONE type = "cell">555-1212</PHONE>
</EMP>
```

Which of the following could NOT be part of a DTD that the document matches? Note: there can be several ATTLIST declarations for a single element; do not assume that there are no attributes for an element other than the one shown in the answer choice.

a) <!ATTLIST EMP ssNo ID #IMPLIED>

b) <!ATTLIST PHONE owner IDREF #REQUIRED>

c) <!ATTLIST EMP name IDREF #REQUIRED>

d) <!ATTLIST PHONE owner CDATA #IMPLIED>

Answer submitted: **b)**

You have answered the question correctly.

Question Explanation:

The correct choices (i.e., the erroneous DTD phrases) are based on two rules:

1. A #REQUIRED attribute must appear in every tag for its element.
2. An attribute can have types CDATA, ID, or IDREF(S), but not #PCDATA.

The incorrect choices (i.e., the phrases that *could* appear in a DTD, are either optional attributes (#IMPLIED) or are required attributes of a proper type.
