5. XPath

W3C fue consciente desde el principio de la necesidad de contar con un modelo de datos común, obviamente basado en la estructura de árbol, sobre el que se dispusiera de un lenguaje capaz de localizar los componentes específicos de un documento XML. Una vez definido se podrían resolver problemas aparentemente tan diversos como:

- Manejar bases de datos de documentos, ya que al interaccionar con la información que contiene cada documento se pueden efectuar las búsquedas correspondientes al manejo de una base documental.
- Dar un marco genérico para poder referenciar, obtener y destacar fragmentos de documentos para su posterior explotación.
- Transformar documentos que en particular servirán para poder aplicar la presentación concreta deseada sobre el resultado de esta transformación.

Se observa que todos estos objetivos tienen en común la necesidad de resolver la localización de componentes de un documento XML, ya que una vez conseguido se pueden construir otras tecnologías y aplicaciones, que permitirán alcanzar los requisitos exigidos a XML en aquellos puntos relacionados con la posibilidad de manipular documentos.

Objetivos

Conocer la recomendación XPath. Entender la recomendación XPath como base de otras tecnologías. Aprender a construir expresiones válidas XPath para sacar información de un documento XML.

Contenidos

5.1. Introducción a XPath

La consecuencia de la necesidad de poder manejar los documentos XML con una cierta solvencia, fue que, solo un año después de sacar la especificación XML, W3C dio a conocer la primera recomendación de XPath (XML Path Language). Es un lenguaje declarativo que proporciona una sintaxis y un modelo de datos para poder localizar y dirigirse a una parte de un documento dado, incorporándole además algunas funcionalidades propias de un lenguaje de propósito general.

Como es conocido, una de las limitaciones de HTML consiste en la dificultad que existe a la hora de seleccionar aquellos fragmentos de un documento, que en su origen no hubieran sido especialmente destacados por el autor. En particular, ello significa que solo se pueden enlazar aquellas partes del documento previamente marcadas al efecto. Ello empezó a ser crítico a medida que la Web crecía, ya que al incrementarse el número de usuarios potenciales de un determinado documento, crece la importancia de poder escoger con libertad las partes a destacar

del mismo. Un simple ejemplo: Obsérvese que al crear cursos *on line*, es básico poder señalar o destacar con libertad una porción o segmento de texto, al igual que se subrayan unos apuntes o un libro.

Para conseguir estas funcionalidades, XPath empezó abordando la posibilidad de localizar los componentes de un documento, siempre con el objetivo de poder destacar fragmentos, tanto de un documento XML como de las entidades externas. Esto se acabaría denominándose un *subrecurso*, que una vez identificado puede a su vez ser objeto de un uso o caracterización especial. XPath es, por tanto, un lenguaje destinado a encaminar hacia una parte del documento para operar sobre él.

A modo de conclusión debe retenerse que gracias a XPath se está en condiciones de poder atravesar el árbol de un documento XML, camino de sus estructuras internas, para un posterior procesado de las mismas.

5.2. Conceptos básicos de XPath

5.2.1. El árbol XPath

Al ser el objetivo principal de XPath dirigirse a una parte de un documento XML, hay que facilitar la posibilidad de moverse a través de él, cosa que se hace utilizando una estructura jerárquica en árbol, y en consecuencia debe contar con un mecanismo que permita al lenguaje seleccionar información del documento objeto del procesamiento. XPath trata a todo documento como un árbol, de forma que siempre cuenta con la capacidad de poder saber si un nodo de este árbol se ajusta o no a la expresión XPath que se utilice en cada momento.

Al objeto de seleccionar partes de un documento, XPath *construye internamente un árbol de nodos* llamado *árbol XPath* (excepto las posibles declaraciones DOCTYPE que no se representan) en el que partir de la raíz, el propio documento, se diversifica a lo largo de los elementos hasta las hojas constituidas bien por valores indivisibles (llamados **átomos**) tales como números, cadenas, etc., o bien por referencias a nodos de otro documento. En coherencia con ello, XPath maneja cuatro tipos de datos: cadenas (de caracteres Unicode), números (en coma flotante), valores booleanos (true o false) y conjuntos de nodos que en la práctica se tratan como una lista. Los nodos de un árbol XPath pueden ser de siete tipos:

- Raíz
- Elemento
- Atributo
- Texto
- Comentario
- Instrucción de Proceso
- Espacio de Nombres.

En este árbol cada nodo intermedio contiene listas ordenadas de nodos hijos, siendo la relación entre un nodo padre y un nodo hijo que el primero contiene al segundo; por ello, además del raíz, solo pueden tener hijos los siguientes nodos: elemento, comentario, texto e instrucción de proceso, mientras que los nodos atributo y espacio de nombres se considera que solo describen a su nodo padre y no contienen nodo alguno.

El uso del término "Path"obedece a que la *estructura* que crea XPath se inspira en el tradicional "File Path" o ruta de acceso utilizada en un disco duro (carpetas y archivos) aunque realmente la *semántica* de XPath sea mucho más parecida a la del SQL en Bases de Datos.

Como veremos, al especificar su sintaxis, XPath utiliza como expresión patrón para identificar los nodos de un documento. La barra (/) al inicio de la expresión, seguida de una lista con los nombres de los elementos hijos que describen un recorrido a través del documento, de forma que tiene capacidad para seleccionar los sucesivos elementos que se ajustan al mismo.

En el sentido semántico, es importante señalar que pesar de que la sintaxis de XPath tenga un estructura similar a la de un *path* en Unix como "/usr/home/pedro/docs", la cual hace referencia a un único archivo *docs* que cuelga del conjunto de directorios /usr/home/pedro, cuando una expresión análoga aparece en XPath, presenta dos diferencias fundamentales respecto a la del Sistema Operativo:

- La primera consiste en que al considerar una expresión XPath como: /libro/capitulo/parrafo, se hace referencia a TODOS los elementos parrafo que cuelguen directamente de CUALQUIER elemento capitulo que cuelgue de CUALQUIER elemento libro, que a su vez cuelgan del nodo raíz.

- La segunda estriba en que XPath no devuelve los elementos que cumplen con el patrón que representa cada expresión, sino que solo devuelve una *referencia*, es decir, una lista de punteros a los elementos que encajan con el patrón de localización. Evidentemente, esta lista resultado puede estar vacía o contener uno o muchos nodos del árbol XPath correspondiente.

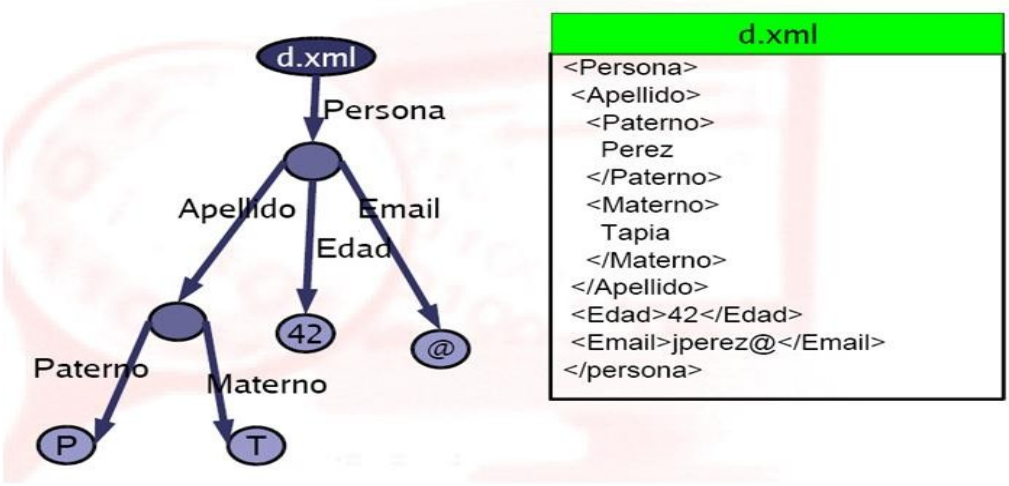
En resumen, en el modelo XPath (que como veremos es compartido por otros lenguajes de la familia XML como XSLT y otros) los datos XML se representan en forma de nodos y valores, que sirven de operandos y de resultados de los operadores. Las sentencias XPath se representan en forma de **secuencias**, entendiendo como tal una colección de uno o más ítems, donde un **ítem** es un nodo o un valor atómico. Por tanto este modelo de datos se basa en 3 bloques:

- **Valores atómicos** de varios tipos, unos generales (cadenas, enteros, etc.) y otros más especializados como nombres cualificados o URI's.

- **Árboles** cuyos nodos representan el contenido de un documento XML

- **Secuencias** (o listas) cuyos ítems son valores atómicos o referencias a nodos en los árboles correspondientes.

De acuerdo con este modelo, cuando se da una expresión XPath aplicada a un documento, su resultado no puede ser otro que una selección de nodos, o un valor atómico (o una sucesión de ellos).



En la imagen anterior podemos ver un ejemplo de árbol XPath construido a partir del documento "d.xml".

5.2.2. Sintaxis y notación

Junto con una biblioteca de funciones, la sintaxis está orientada tanto a definir partes del documento como a proporcionar rutas hacia los elementos. En consecuencia, XPath utiliza la barra (/) seguida de una lista con los nombres de los elementos hijo, que en su conjunto describen un recorrido, a través del documento, de forma que selecciona los sucesivos elementos que se ajustan al recorrido expresado por la secuencia como expresión patrón para identificar los nodos de un documento, Así, dado el siguiente documento con la lista de platos internacionales de un restaurante:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<comidas>
        <plato pais="Italia">
                <nombre>Lassagna </nombre>
                <precio>9.90</precio>
        </plato>
        <plato pais="España">
                <nombre>Paella de marisco</nombre>
                <precio>10.50</precio>
        </plato>
        <plato pais="Colombia">
```

```
            <nombre>Bandeja paisa</nombre>
            <precio>10.50</precio>
        </plato>
        <plato pais="Mexico">
            .........
        </plato>
        ........
</comidas>
```

Cuando se escriba la secuencia: “comidas/plato/precio”, en el marco adecuado, XPath acabará seleccionando todos los precios de todos los platos que figuran en él. El resultado de la expresión anterior sería la siguiente:

```
<precio>9.90</precio>
<precio>10.50</precio>
<precio>10.50</precio>
........
```

La colección de todos los nodos precio hijos de los nodos plato que a su vez son hijos del nodo comidas.

5.2.3. Trayecto de búsqueda

En XPath se llama *trayecto de búsqueda* (*location path)* al conjunto de nodos sobre los que se evalúa una expresión XPath, que obtendrá como resultado otro conjunto de nodos, formado por aquellos que cumplen los criterios especificados. Para dar los nodos del trayecto, se empieza designando un punto concreto, llamado *nodo contexto*, que es el nodo del árbol XPath del documento desde el que se inicia el trayecto de búsqueda.

Tanto el concepto de *nodo contexto* como *trayecto de búsqueda* son análogos a los utilizados en la gestión de archivos. De esta forma, en XPath, a menos que se indique un trayecto explícito se entiende que el trayecto de búsqueda parte del nodo que en cada momento se esté procesando. Por ello, los trayectos de búsqueda pueden ser *absolutos* o *relativos*; los primeros empiezan con una barra (/) que determina el nodo contexto, mientras que uno relativo carece de ella al inicio, y depende del lugar del documento en el que se esté trabajando al invocar el trayecto de búsqueda.

Sea absoluto o relativo, un trayecto consiste en una serie de escalones separados por una barra, de forma que un trayecto de búsqueda absoluto es de la forma: “/escalon/escalon/...” y uno relativo adopta la formula: “escalon/escalon/... “.

La expresión de un trayecto se evalúa de izquierda a derecha. Así, dado el documento siguiente con la estructura habitual de un libro:

```
<Libro>
    <Capitulo numero="1">
        <Parrafo numero="1"> texto. </Parrafo>
        <Parrafo numeror="2"> texto. </Parrafo>
```

```
            .....
        </Capitulo>
        <Capitulo numero="2">
            <Parrafo numero="1"> texto. </Parrafo>
            <Parrafo numero="2"> texto. </Parrafo>
              …..
        </Capitulo> …..
</Libro>
```

La expresión: "/libro/capitulo/parrafo", al empezar con /, selecciona el nodo raíz, independientemente del nodo contexto de cada momento; a continuación al leer *libro*, se seleccionan TODOS los elementos que cuelgan del nodo contexto (que en este caso es único por ser raíz), al seguir y leer capitulo se seleccionan TODOS los elementos que cuelgan del nodo contexto. Al llegar a este punto hay que hacer notar que si se estuviera gestionando un disco sería imposible que hubiera dos directorios con el mismo nombre colgando de un mismo directorio padre, cosa que sin embargo, en un documento XML es perfectamente normal, como en el ejemplo donde son varios los hijos *capitulo* que cuelgan del elemento libro. Al continuar leyendo se llega al elemento <*parrafo*>, con lo que se indica seleccionar TODOS los elementos <*parrafo*> que cuelgan del nodo contexto (ahora capitulo) aunque haya varios nodos contexto.

Ante este tipo de situaciones, el evaluador de expresiones XPath los va a recorrer uno por uno, siguiendo la regla de "primero en profundidad" (empezar por la primera rama hasta llegar a un nodo hoja y volver para seguir por la rama siguiente para proceder de igual forma hasta agotar todas las posibilidades) mientras se evalúa un determinado nodo, éste sea el nodo contexto momentáneo.

Como se adelantó, el resultado al evaluar una expresión XPath es un conjunto de punteros a nodos que encajan con el patrón buscado. Aunque no es nuestro objetivo profundizar en las particularidades de la sintaxis de XPath, ya hemos podido ver que la sintaxis habitual para tratar un árbol incorpora una serie de abreviaciones que son muy evidentes y que vale la pena citar:

- En un trayecto de búsqueda, cuando el nodo raíz es el nodo contexto, basta con escribir "/" .
- Una expresión como *child::Autor* se abrevia como Autor (como veremos el llamado eje child es el eje por defecto).
- El operador ("//") indica que se seleccionen todos los elementos del documento que cumplan los criterios con independencia del nivel que ocupen en el árbol. Por ejemplo: "//plato" selecciona todos los elementos plato de la lista de comidas.
- El operador ("|") selecciona varios recorridos; por ejemplo: "/comidas/plato/nombre | /comidas/plato/precio" proporciona todos los elementos nombre y elemento precio de los platos del documento.

Las abreviaturas anteriores, se pueden combinar y así:

- //nombre | //precio,

Selecciona todos los elementos nombre y precio del documento.

- /comidas/plato/nombre | //precio

Selecciona todos los elementos nombre del elemento plato del elemento comidas, así como todos los elementos precio del documento.

5.3. Elementos de un trayecto de búsqueda

En todo trayecto de búsqueda de XPath se pueden distinguir tres elementos básicos llamados Ejes, Nodos y Predicados. Siendo esta sintaxis de localización en XPath:

```
nombreeje::nodocomprobacion[predicado]
```

Como por ejemplo (Más delante veremos lo que significa):

```
child::precio[precio=9.90]
```

5.3.1. Ejes

Los Ejes (axes) son elementos encargados de especificar en un trayecto de búsqueda la relación existente dentro del árbol XPath, entre los nodos que quieren seleccionarse en el trayecto, y el nodo contexto de donde se parte. El término inglés *axes* significa también, además del plural de eje, "cercenar" o "podar" y ambos conceptos serían adecuados, ya que con el eje incluido en un trayecto se realiza una selección de nodos de acuerdo con el patrón, dentro del árbol o mejor dicho, dentro del subárbol que cuelga del nodo contexto. Se observa, en los ejemplo vistos, que el uso de la barra / (salvo cuando ha servido para denominar el nodo raíz) determina un eje.

En XPath, existen 13 tipos de ejes siendo los más habituales son:

Self: Identifica el nodo contexto y se especifica mediante un punto (.)

Child: Describe los hijos del nodo contexto, y se expresa de la forma : "/child::", aunque como se ha indicado "child" puede omitirse de un trayecto de búsqueda, al ser el eje por defecto y así para seleccionar los títulos de libro basta con usar: /libro/titulo.

Descendant: Selecciona cualquier nodo que sea descendiente del conjunto de nodos contexto, expresándose de la forma: "/descendant::", palabra que puede evitarse pues siempre se asume por defecto tras //. Para seleccionar todos los párrafos de libro, se escribe**:** /libro//parrafo.

Attribute: Contiene los atributos de un determinado nodo y se abrevia usando @ como prefijo del elemento buscado; por ejemplo: "//@pais" selecciona todos los atributos con el nombre de pais, aplicado al documento de platos.

Parent: Indica el nodo padre del nodo contexto y se identifica con dos puntos seguidos(..). Así, para seleccionar los nodos que tienen algún hijo de parrafo escribiríamos: "//parrafo/.." mientras que para seleccionar los nodos capitulo que tienen algún hijo de tipo párrafo se escribe: "//capitulo/parrafo/.."

5.3.2. Nodos de comprobación o búsqueda

Los nodos de comprobación o búsqueda (*node test*) son los encargados de identificar un nodo o nodos concretos dentro de un eje, siendo su función dentro del trayecto de búsqueda que cada eje pueda determinar un tipo de nodo (llamado *principal*) a la hora de efectuar la selección. Este nodo de comprobación puede indicarse bien por nombre o bien por tipo.

Son los nodos más importantes y usados:

node( ): Devuelve todos los nodos de cualquier tipo. Así para seleccionar todos los nodos descendientes del tipo párrafo, se escribe: "/parrafo/node( )"

***** (llamado *wildcard* siguiendo la jerga del tenis): Selecciona los nodos principales de cada trayecto (a excepción de los tipo: texto, comentario e instrucciones de proceso) indicando su nivel en el árbol. Así:

•"/comidas/plato/*" selecciona todos los elementos hijos de todos los plato.

•"/catalogo/*/precio" obtiene los precios de los elementos que son nietos del elemento catálogo.

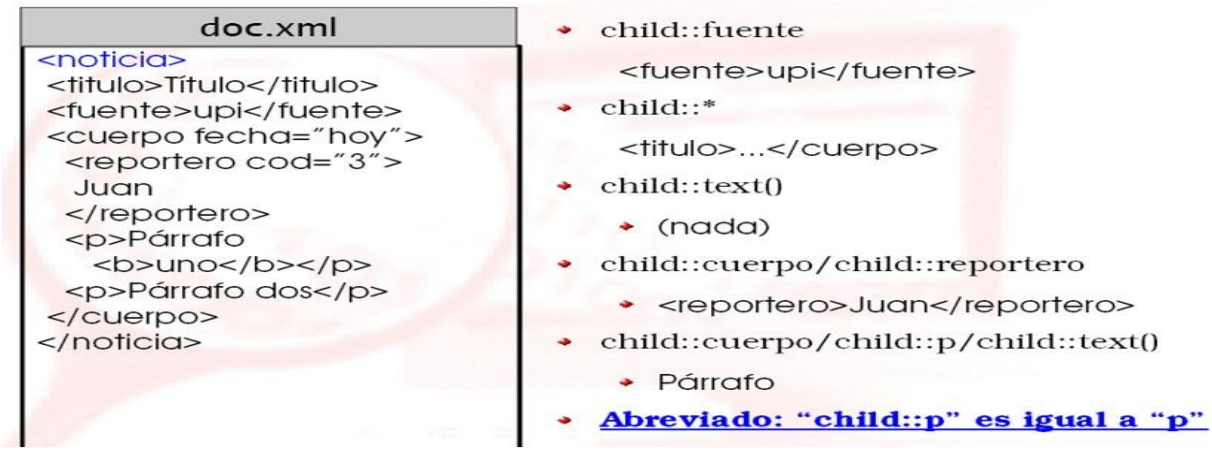
•"/*/*/precio" obtiene el precio de los elementos que tienen dos antecesores.

•"//*" selecciona todos los elementos del documento.

text ( ): Devuelve cualquier nodo de tipo texto. Así para seleccionar el texto de todos los nodos párrafo se escribe: "//parrafo/text( )" y para seleccionar TODO el texto que cuelga de todos los nodos tipo párrafo: "//parrafo//text( )"

5.3.3. Algunos ejemplos

En las imágenes siguientes podemos ver algunos ejemplos de expresiones XPath que utilizan lo visto hasta ahora. Cada imagen se compone de un fichero XML sobre el que se aplican una serie de expresiones XPath y vemos el resultado obtenido. En los datos XML veremos siempre algún nodo resaltado en azul, esto significa que es el nodo de contexto.



doc.xml

```
<noticia>
 <titulo>Título</titulo>
 <fuente>upi</fuente>
 <cuerpo fecha="hoy">
  <reportero cod="3">
   Juan
  </reportero>
  <p>Párrafo
    <b>uno</b></p>
  <p>Párrafo dos</p>
 </cuerpo>
</noticia>
```

- parent::cuerpo/parent::noticia
  - <noticia>...</noticia>
- **Abreviado: "parent::*" es igual a ".."**
- ../../fuente
  - <fuente>upi</fuente>
- ancestor::noticia/titulo
  - <titulo>Título<titulo>
- self::reportero
  - <reportero>Juan</reportero>
- **Abreviado: "self::*" es igual a "."**

doc.xml

```
<noticia>
 <titulo>Título</titulo>
 <fuente>upi</fuente>
 <cuerpo fecha="hoy">
  <reportero cod="3">
   Juan
  </reportero>
  <p>Párrafo
    <b>uno</b></p>
  <p>Párrafo dos</p>
 </cuerpo>
</noticia>
```

- attribute::fecha
  - hoy
- **Abreviado: "attribute::x" es igual a "@x"**
- reportero/@cod
  - 3
- descendant::b
  - uno
- **Abreviado: "descendant::b" igual a ".//b"**
- .//noticia
  - <noticia>...</noticia>
- p/b/text()
  - uno

doc.xml

```
<noticia>
 <titulo>Título</titulo>
 <fuente>upi</fuente>
 <cuerpo fecha="hoy">
  <reportero cod="3">
   Juan
  </reportero>
  <p>Párrafo
    <b>uno</b></p>
  <p>Párrafo dos</p>
 </cuerpo>
</noticia>
```

- preceding::*
  - <reportero cod="3">Juan </reportero>
- following::p
  - Párrafo dos

5.3.4. Predicados

Los predicados (predicates) son elementos que en un trayecto de búsqueda permiten restringir el conjunto de nodos seleccionados por un eje, a aquellos que cumplan una cierta condición, de forma que un predicado especifica con mayor detalle la información que se requiere, permitiendo filtrar un conjunto dado. Para la obtención de un predicado se recurre a identificaciones o a las funciones propias de XPath que veremos a continuación escritas entre corchetes ([....]) .Son ejemplos de predicados:

- "/comidas/plato[1]" selecciona el primer plato de comidas.
- "comidas/plato[last()]" lo hace con el último (nótese que no existe la función first() ya que para ello se usa un predicado como plato[1]).
- "/comidas/plato[precio]" obtiene los elementos que tengan elemento precio.
- "comidas/plato[precio=10.50]/precio", obtiene los precios de los platos con un precio cuyo valor sea 10.50.
- "//*[@num]" selecciona todos los elementos que tengan el atributo num .
- "//capitulo[parrafo/*[@href]]" selecciona todos los capítulos que tengan un párrafo que a su vez tenga algún elemento con atributo href.
- /libro/capitulo[@num="1"]/parrafo, que escoge aquellos elementos parrafo de todos los elementos capitulo que tengan un atributo, llamado num cuyo valor sea 1.

Es importante señalar que los predicados pueden sucederse uno a otro, cosa que tiene el efecto de la operación lógica "Y". Así para seleccionar todos los capítulos que tengan un párrafo que tenga algún elemento con atributo href y a su vez capitulo tenga el atributo public con valor si se puede escribir de la forma:

```
"//capitulo[parrafo/*[@href]][@public='si']"
```

Una conclusión relevante de lo visto es que el uso tanto de nodos de comprobación como de los predicados pone de manifiesto que XPath es más que un mero mecanismo de selección de varios nodos a la vez, ya que como se ha visto, este lenguaje tiene la capacidad de individualizar un nodo con características definidas, especialmente mediante predicados incluidos dentro del trayecto.

5.4. Funciones XPath

Para que XPath resulte más poderoso y útil existe una gran cantidad de funciones que se pueden usar en las expresiones XPath. Algunas de estas funciones se pueden utilizar para retornar nodos y conjuntos de nodos que no pueden ser encontrados por medio de las relaciones normales de hijo/padre y elemento/atributo. También existen funciones para utilizar con cadenas y números, que se pueden usar para recuperar información desde el documento original y para formatearla para la salida.

Las funciones se pueden reconocer fácilmente porque siempre terminan con "()". Algunas funciones necesitan información para funcionar, esta información se pasa en forma de parámetros.

Por ejemplo veremos más adelante una función para cadenas "string-length()", que devuelve la cantidad de caracteres de una cadena. Puede utilizarse de la siguiente manera:

```
String-length('Esta es una cadena')
```

El parámetro es la cadena "Esta es una cadena". La función usará esta información para calcular el resultado que en este caso será 18.

5.4.1. Funciones de nodo

name(). La función name() se usa para obtener el nombre de un nodo. Por ejem plo name(.) devuelve el nombre del nodo de contexto. Además '.' Es el parámetro por defecto por lo cual name()=name(.).

node(). Retorna el propio nodo. No se suele utilizar.

procesing-instruction(). Para retornar instrucciones de procesamiento. Esta función utiliza un parámetro opcional para especificar el nombre.

comment(). La función comment() se usa para devolver comentarios. Hay que tener en cuenta que existen *parsers* que no pasan los comentarios y por lo tanto esta función no devolverá nunca ningún valor.

text(). Esta función devuelve el contenido PCDATA de un nodo, sin el PCDATA de ningún hijo, si es que hubiera alguno. Por ejemplo: "/comidas/plato[1]/nombre/text()" devuelve: 'Lassagna'.

5.4.2. Funciones posicionales

position(). Se utiliza para obtener la posición del nodo en un conjunto de nodos. Por ejemplo si tenemos el siguiente ejemplo:

```
<nodes>
<node>a</node>
<node>b</node>
<node>c</node>
</nodes>
```

Si queremos el Segundo nodo debemos especificarlo de la siguiente manera: "/nodes/node[position()=2]" esto nos devolverá "<node>b</node>". Como esta función se utiliza muy a menudo se puede abreviar "/nodes/node[2]" nos devuelve el mismo resultado.

last(). Esta función nos devuelve la posición del último nodo en un conjunto de nodos. En el ejemplo anterior "/nodes/node[position()=last()] nos devolverá "<node>c</node>". No existe una función *first()* ya que es equivalente a "position()=1".

count(). Retorna la cantidad de nodos en un conjunto. Por ejemplo, para obtener la cantidad de elementos *node* del XML anterior: "count(/nodes/node)".

5.4.3. Funciones numéricas

number(). La función convierte texto PCDATA en un valor numérico. Por ejemplo si suponemos que tenemos el siguiente elemento:" <element>256</element>". Para XPath el valor del elemento es una cadena que contiene los caracteres '2', '5', '6'. Para poder tratar el valor como un número debemos utilizar la función: "number(element)".

sum(). Se utiliza para unir todos los valores numéricos en un conjunto de nodos. Por ejemplo si

cambianos nuestro ejemplo XML anterior por:

```
<nodes>
<node>1</node>
<node>2</node>
<node>3</node>
</nodes>
```

Podemos sumar el valor total de nodes con la siguiente expresión: “sum(/nodes/node)”. Si no hubiésemos cambiado el documento XML y aplicamos la expresión al original, la función nos devolvería ‘NaN’ (*not a number*). Ya que XPath no es capaz de convertir ‘a’, ‘b’, y ‘c’ a números.

5.4.4. Funciones Booleanas

boolean(). Simplemente evalúa una expresión para verificar si es verdadero o falso utilizando las siguientes reglas:

- Si el valor es numérico se considera falso si es 0 o el valor especial NaN. Si el número tiene cualquier otro valor (positivo o negativo) se considera verdadero.
- Si el valor es una cadena se considera verdadero si su longitud es mayor que 0.
- Si el valor es un conjunto de nodos, se considera verdadero si la colección no está vacía.
- Cualquier otro tipo de objeto se convierte a booleano dependiendo del tipo de objeto.

Por ejemplo la expresión “boolean(name)” se evaluará verdadera si existe un nodo *name* hijo del nodo contexto.

not(). Para poder hacer algo cuando se cumple lo contrario de nuestra expresión. Por ejemplo si en la expresión anterior queremos hacer algo cuando no existe no un nodo hijo *name*, la expresión a utilizar sería: “not(boolean(name))”.

true() y false(). XPath nos proporciona estas dos funciones que no admiten parámetros y que siempre devuelven verdadero o falso respectivamente.

5.4.5. Funciones de cadena

Antes de comenzar a ver las distintas funciones resaltar que las funciones de cadena en XPath consideran de manera distinta las mayúsculas y las minúsculas.

string(). La función convierte cualquier valor a cadena. Normalmente la función no es necesaria al leer los datos desde el árbol origen, ya que los datos están todos en formato de texto. Puede ser útil por ejemplo para convertir resultados de cálculos en cadenas de texto.

string-length(). La función devuelve la cantidad de caracteres de la cadena pasada como parámetro incluidos los espacios.

concat(). La función toma como parámetros varias cadenas y devuelve el resultado de concatenarlas. Por ejemplo “concat(‘esto’, ‘ es ’, ‘ una ‘, ‘ cadena’)” devolvería la cadena ‘esto es una cadena’. Puede ser útil por ejemplo para concatenar distintos CDATA.

contains(). La función indica si una cadena contiene dentro de si misma otra cadena. Por ejemplo “contains(‘Esto es una cadena’, ‘es una’)” devuelve verdadero.

stars-with(). La función devuelve verdadero si la primera cadena comienza con la segunda cadena pasada como parámetro. "stars-with('Esto es una cadena', 'Esto')" devolvería verdadero.

substring(). La funciónutiliza tres parámetros. La cadena de la que se van a extraer los caracteres, la posición donde se empieza a tomar los caracteres y por último la cantidad de caracteres que se toman. EL último parámetro es opcional y si se omite se toman todos hasta el final de la cadena. Solo hay que tener en cuenta que a diferencia de en muchos lenguajes de programación en XPath el primer elemento de una cadena es el 1. Por ejemplo la expresión "substring('Esto es la cadena principal', 12)" devuelve 'cadena principal'. Al omitir el tercer parámetro a tomado desde el carácter 12 (inclusive) hasta el final de la cadena.

substring-after(). La función retorna todos los caracteres de la cadena que están después del carácter pasado como segundo parámetro. Por ejemplo "substring-after('Esto es', 't')" devuelve la cadena 'o es'.

substring-before(). La función devuelve al contrario de la anterior el texto anterior al carácter pasado como parámetro. Por ejemplo "substring-before('Esto es', 't')" devuelve la cadena 'Es'.

translate(). La función resulta un poco más complicada de lo que puede parecer y lo mejor es ver algunos ejemplos:

```
translate('12:30', '30', '45')→ devuelve '12:45'
translate('12:30', '03', '54')→ devuelve '12:45'
```

Si nos fijamos las dos devuelven el mismo resultado. Lo que realmente hace la función es tomar de los dos últimos parámetros los caracteres por separado y cambia el primero del segundo parámetro por el primero del tercero, el segundo del segundo parámetro por el segundo del tercero etc. De esta manera en el primero ejemplo le estamos diciendo cambia el 3 por un 4 y el 0 por un 5, y en el segundo le decimos cambial el 0 por un 5 y el 3 por un 4 que es lo mismo que en el primer ejemplo pero en orden inverso. Con el siguiente ejemplo puede quedar un poco más claro:

```
translate('12:30','0123', 'abcd')→ devuelve 'bc:da'
```

Podemos utilizar la función para convertir de mayúsculas a minúsculas oviceversa:

```
translate('convertir', 'abcdefghijklmnopqrstuvwxyz', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ')
translate('CONVERTIR', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ', 'abcdefghijklmnopqrstuvwxyz')
```

Solo hay que tener en cuenta dos cosas más. Si un carácter no está en el segundo parámetro quedará como en el original: "translate('aBc', 'abc','012')" devuelve '0B2' ya que B no tiene correspondencia en el segundo parámetro.

Por otro lado si el segundo parámetro es más largo que el tercero todos los caracteres del segundo parámetro que no tienen correspondencia en el tercero desaparecen, por ejemplo "translate('aBc', 'abcB', '012')" devuelve '02'. El carácter 'B' desaparece ya que está en el segundo parámetro pero no tiene correspondencia en el tercero.

5.4.6. Algunos ejemplos

libros.xml

```
<libros>
 <libro>
  <titulo>XXX</titulo>
  <año>1890</año>
 </libro>
 <libro>
  <titulo>YYY</titulo>
  <año>1950</año>
 </libro>
 <libro>
  <año>1830</año>
 <libro>
</libros>
```

- libro[titulo = 'XXX']/año
  - 1890
- libro[not(titulo = 'XXX')]
  - 1950

libros.xml

```
<libros>
 <libro>
  <titulo>XXX</titulo>
  <año>1890</año>
 </libro>
 <libro>
  <titulo>YYY</titulo>
  <año>1950</año>
 </libro>
 <libro>
  <año>1830</año>
 <libro>
</libros>
```

- concat(libro[1]/titulo, libro[2]/año)
  - XXX1950
- libro[starts-with (titulo,'X')]/año
  - 1890
- libro[**contains**(año,9)]/ año
  - 1950

libros.xml

```
<libros>
 <libro>
  <titulo>XXX</titulo>
  <año>1890</año>
 </libro>
 <libro>
  <titulo>YYY</titulo>
  <año>1950</año>
 </libro>
 <libro>
  <año>1830</año>
 <libro>
</libros>
```

- libro[position()=last()]/año
  - 1830
- count(libro)
  - 3
- libro[count(titulo)=0]/ año
  - 1830
- count(libro/titulo)
  - 2

5.5. Utilidades de XPath

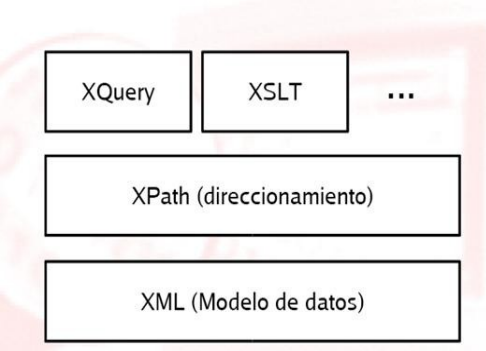
Como conclusión vamos a tratar de explicar cuál es la utilidad de XPath dentro del mundo XML. Bien podemos hacer un símil entre la sentencia SELECT de SQL y XPath. Normalmente en la programación estándar de hoy en día se trabaja contra una base de datos relacional. Independientemente del lenguaje de programación utilizamos sentencias SELECT para recuperar información de la base de datos según el programa la va necesitando. Con XPath y XML pasa lo mismo. XPath no es el *fin* sino el *medio*.

Cuando trabajemos con ficheros XML utilizaremos diferentes lenguajes XML (XSLT, XQUERY, etc.) o de programación (JAVA) que utilizarán XPath para poder seleccionar información del fichero XML para poder trabajar sobre ella.

Pero a menudo también utilizamos SQL contra una base de datos fuera de un lenguaje de programación. Bien sea para obtener información inmediata, bien para probar sentencias SQL que luego incorporaremos a nuestra base de datos. Con XPath igual, podemos necesitar en algún momento extraer cierta información de un documento XML o probar diferentes expresiones para luego incorporarlas a alguno de nuestros programas.

También estaremos de acuerdo en que para crear o modificar aplicaciones de gestión (que conecten con bases de datos relacionales) es indispensable conocer SQL. Cuanto mejor conozcamos SQL más fiables y rápidos serán nuestros programas. Por ejemplo, una persona que conozca la sentencia SELECT como 'SELECT * FROM' y no conozca el WHERE o el ORDER BY, estaremos de acuerdo en que por muy bien que pueda programar los programas serán pobres y difíciles de mantener. Ya que, siempre tendrá que filtrar y ordenar en el lado cliente.

Con XPath igual, cuanto mejor lo conozcamos más eficientes serán nuestras aplicaciones y más sencillas de mantener. Si podemos evitar recorrer un árbol XML entero extrayendo solo la información que necesitamos en cada momento menos trabajo de programación, menos líneas de código, menos probabilidad de error en el programa.

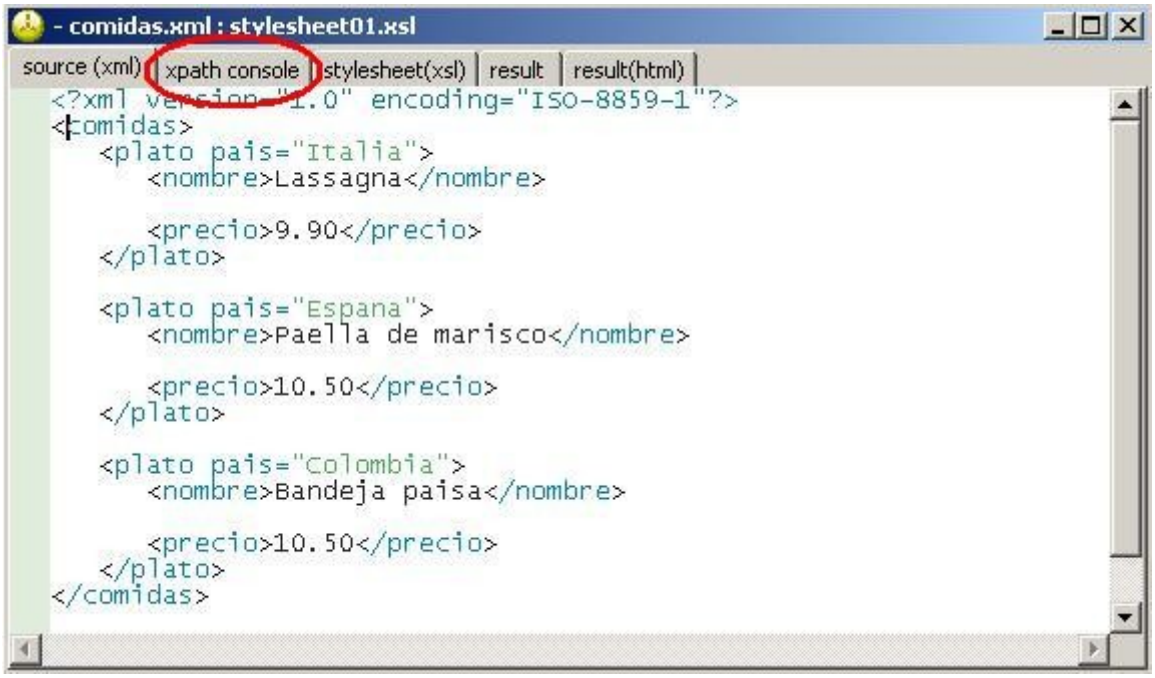


Como podemos ver en la imagen de la izquierda XPath es el camino que utilizan otros lenguajes para obtener información del fichero XML.

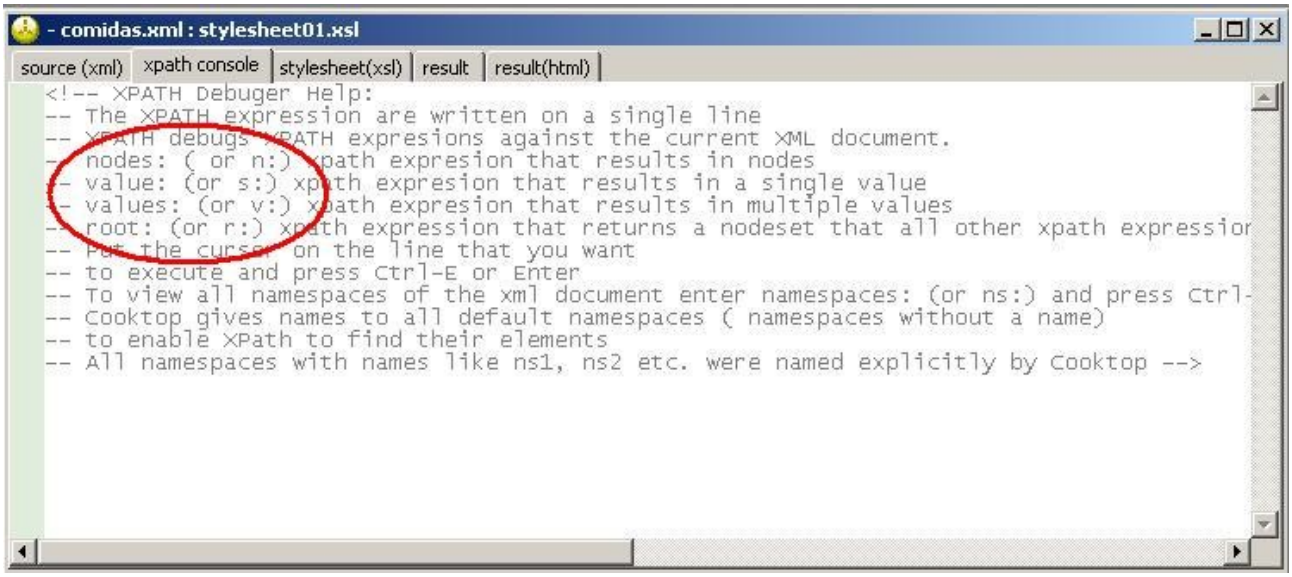
En el siguiente tema veremos XSLT y veremos cómo este lenguaje utiliza XPath para obtener información del fichero XML para construir un nuevo fichero con un formato diferente.

5.6. Probar expresiones XPath

Para probar expresiones XPath vamos a utilizar la aplicación *Cooktop* que ya conocemos de temas anteriores y debemos tener instalada en el ordenador. Vamos a trabajar sobre el fichero '*comidas.xml'*. Podemos copiarlo directamente (es el XML de ejemplo utilizado en este mismo capítulo en el apartado 5.2.2).



Una vez tenemos cargado y verificado el fichero XML podemos ir a la consola XPath (en la imagen anterior podemos verla con un círculo rojo). Al pulsar sobre esta opción nos aparece la siguiente información:



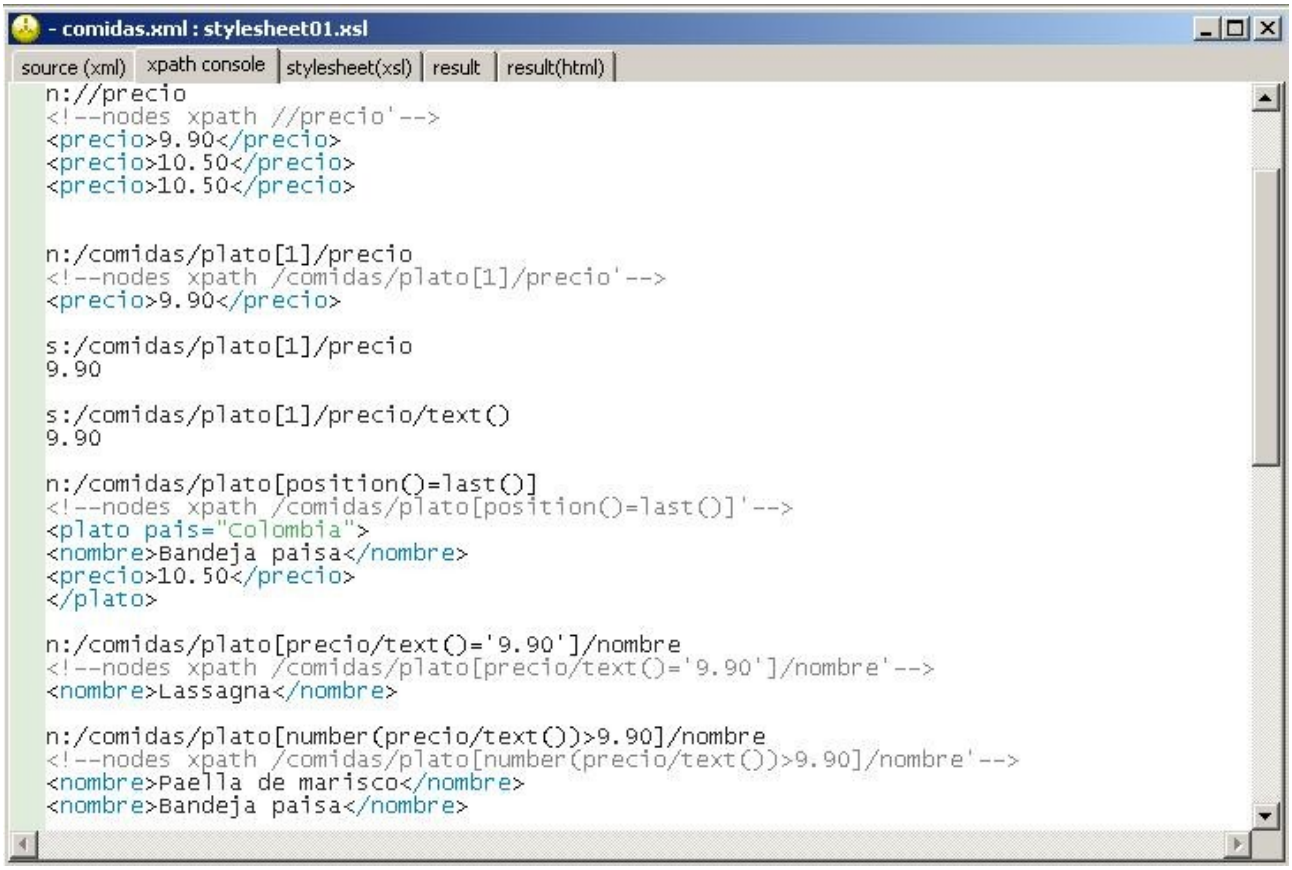
Al ir a este apartado *Cooktop* nos muestra información de cómo debemos utilizar esta herramienta. Lo más importante es que dependiendo del resultado que preveemos obtener de la expresión XPath debemos utilizar:

- nodes: (o n:) cuando esperamos obtener una lista de nodos.
- value: (o s:) cuando esperamos obtener un valor.
- values: (o v:) cuando esperamos obtener una lista de valores

- root: (o r:) cuando queremos cambiar el nodo de contexto sobre el que trabajarán las siguientes expresiones XPath.

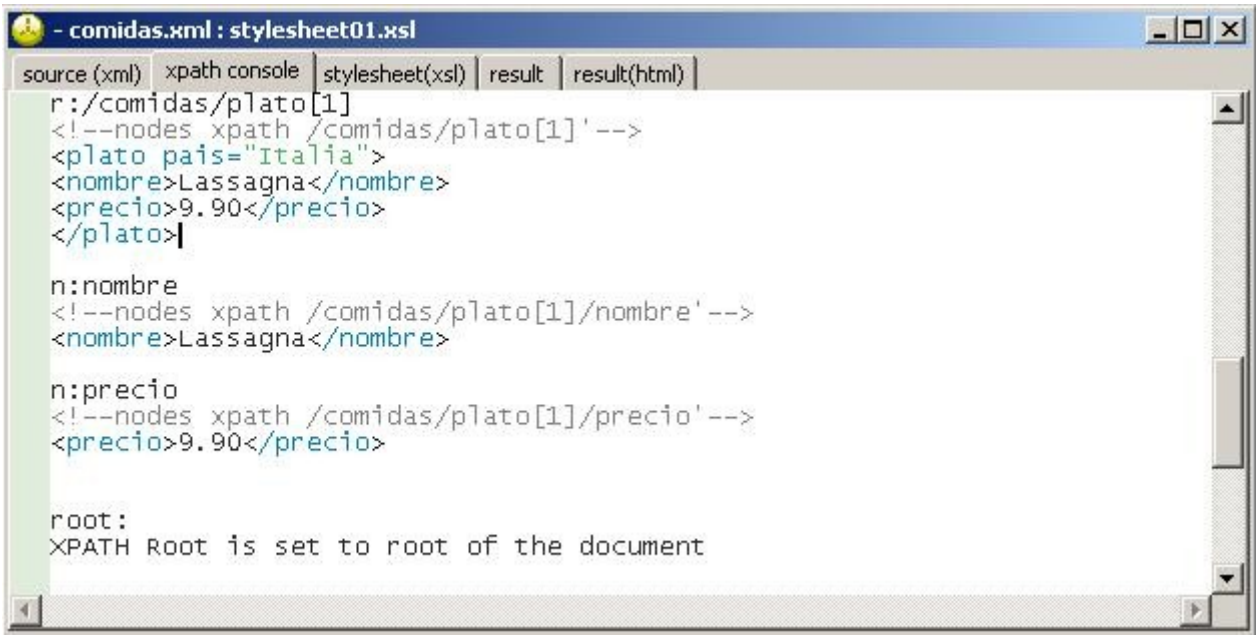
Por ejemplo, imaginemos que queremos sacar el precio de todas las comidas existentes en el fichero comidas XML. La expresión XPath es "/comidas/plato/precio" o "//precio". Como lo que esperamos que devuelve es un conjunto de nodos escribiremos la siguiente orden en la consola: "n:/comidas/plato/precio" y pulsaremos [enter]. El resultado debe ser el siguiente:

Podemos ver en la imagen que nos devuelve los tres nodos esperados, de los tres platos existentes en el fichero XML. Siguiendo este ejemplo podemos ver la siguiente imagen con algunos ejemplos más que podemos probar:



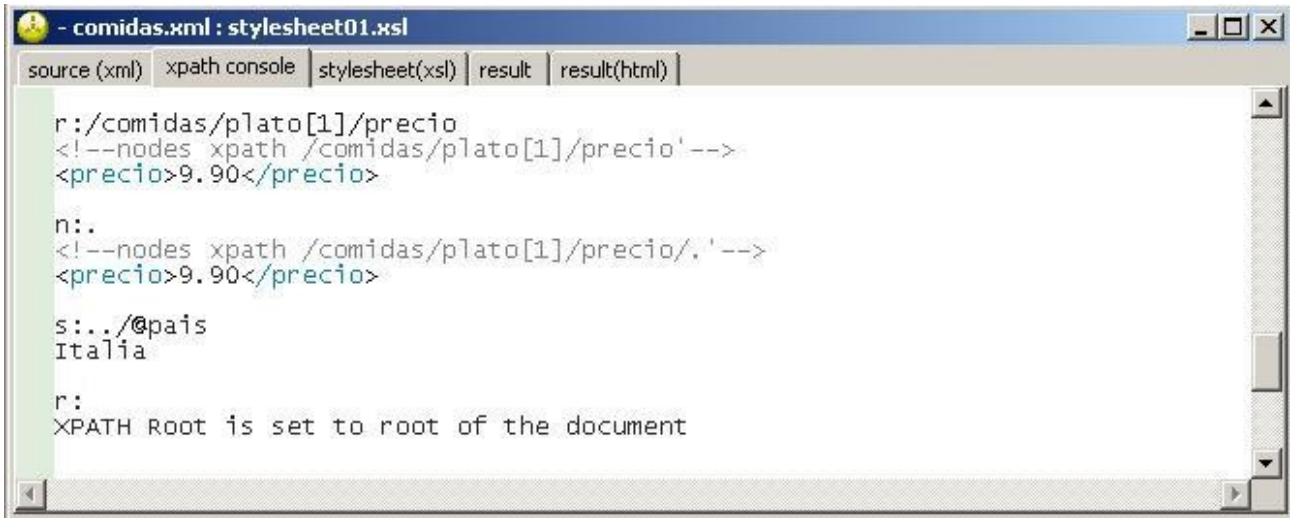
Ahora es el momento de echar un poco de imaginación y realizar las pruebas que se nos ocurran con las diferentes funciones que hemos visto en el tema.

Un último ejemplo un poco más complicado de entender es la utilización del comando 'r:' de *Cooktop*. En la imagen siguiente podemos verlo:



Con la primera orden "r:/comidas/plato[1]" hemos cambiado el nodo de contexto al primer plato. Ahora las siguientes órdenes trabajarán sobre este nodo de contexto. La siguiente orden "n:plato" que inicialmente no hubiese dado ningún resultado ahora sí, ya que como podemos ver *Cooktop* aplica la expresión sobre el nuevo *root* definido anteriormente. De esta manera podemos cambiar el nodo de contexto tantas veces como queramos. Por último con la orden "r:" o "root:" tal y como vemos al final de la imagen devolvemos el nodo de contexto al nodo raíz del documento XML.

Por ejemplo imaginemos que queremos nuestro nodo de contexto es el precio del primer plato. Tendremos que ejecutar la siguiente orden "r:/comidas/plato[1]/precio". Ahora desde este nodo de contexto queremos saber el valor del parámetro país del elemento plato que es padre del precio que tenemos como nodo de contexto. La expresión será "s:../@precio". Veámoslo ahora en la siguiente imagen de *Cooktop*:



Con la primera orden cambiamos el nodo de contexto. Con la siguiente consultamos en nod actual (que debe ser el de contexto). Con la siguiente seleccionamos el atributo país del padre del elemento que tenemos como nodo de contexto. Y por último devolvemos el nodo de contexto a la raíz del documento XML.

5.7. Ejercicios

Utiliza la notación abreviada de XPath vista para seleccionar lo que se especifica en los siguientes ejercicios. Debes mostrar también su resultado. Para evitar copiar de nuevo los trozos de código XML, el resultado lo podéis señalar por ejemplo mediante un círculo o subrayándolo con color. A continuación mostramos un ejemplo de cómo debéis presentar los resultados para este primer bloque.

Nota: Usar caminos absolutos para todos los ejercicios.

5.7.1. Ejercicio 1. (Ejemplo)

Selecciona el elemento raíz AAA.

```
<AAA>
    <BBB/>
    <CCC/>
    <BBB/>
    <BBB/>
    <DDD>
        <BBB/>
    </DDD>
    <CCC/>
</AAA>
```

SOLUCIÓN:

Notación abreviada a utilizar de XPath sería: "/AAA"

Y el resultado está marcado con color amarillo.

5.7.2. Ejercicio 2

Selecciona todos los elementos CCC que son hijos del elemento raíz AAA

```
<AAA>
    <BBB/>
    <CCC/>
    <BBB/>
    <BBB/>
    <DDD>
        <BBB/>
    </DDD>
    <CCC/>
</AAA>
```

5.7.3. Ejercicio 3

Selecciona todos los elementos BBB que son hijos de DDD, que a su vez son hijos del elemento raíz AAA.

```
<AAA>
        <BBB/>
        <CCC/>
        <BBB/>
        <BBB/>
        <DDD>
            <BBB/>
        </DDD>
        <CCC/>
  </AAA>
```

5.7.4. Ejercicio 4

Selecciona todos los elementos BBB

```
<AAA>
        <BBB/>
        <CCC/>
        <BBB/>
        <DDD>
            <BBB/>
        </DDD>
        <CCC>
            <DDD>
                <BBB/>
                <BBB/>
            </DDD>
        </CCC>
 </AAA>
```

5.7.5. Ejercicio 5

Selecciona todos los elementos BBB que son hijos de DDD

```
<AAA>
      <BBB/>
      <CCC/>
      <BBB/>
      <DDD>
         <BBB/>
      </DDD>
      <CCC>
         <DDD>
            <BBB/>
            <BBB/>
         </DDD>
      </CCC>
</AAA>
```

5.7.6. Ejercicio 6

Selecciona todos los elementos contenidos en el camino /AAA/CCC/DDD

```
<AAA>
      <XXX>
         <DDD>
            <BBB/>
            <BBB/>
            <EEE/>
            <FFF/>
         </DDD>
      </XXX>
      <CCC>
         <DDD>
            <BBB/>
            <BBB/>
            <EEE/>
            <FFF/>
         </DDD>
      </CCC>
      <CCC>
         <BBB>
            <BBB>
               <BBB/>
            </BBB>
         </BBB>
      </CCC>
  </AAA>
```

5.7.7. Ejercicio 7

Selecciona todos los elementos BBB incluidos en exactamente 3 antecesores

```
<AAA>
      <XXX>
         <DDD>
            <BBB/>
            <BBB/>
            <EEE/>
            <FFF/>
         </DDD>
      </XXX>
      <CCC>
         <DDD>
            <BBB/>
            <BBB/>
            <EEE/>
            <FFF/>
         </DDD>
      </CCC>
      <CCC>
         <BBB>
            <BBB>
               <BBB/>
            </BBB>
         </BBB>
      </CCC>
  </AAA>
```

5.7.8. Ejercicio 8

Selecciona todos los elementos

```
<AAA>
      <XXX>
         <DDD>
            <BBB/>
            <BBB/>
            <EEE/>
            <FFF/>
         </DDD>
      </XXX>
      <CCC>
         <DDD>
            <BBB/>
            <BBB/>
            <EEE/>
            <FFF/>
         </DDD>
      </CCC>
      <CCC>
         <BBB>
            <BBB>
               <BBB/>
            </BBB>
         </BBB>
      </CCC>
   </AAA>
```

5.7.9. Ejercicio 9

Selecciona el primer hijo BBB del elemento AAA

```
<AAA>
      <BBB/>
      <BBB/>
      <BBB/>
      <BBB/>
</AAA>
```

5.7.10. Ejercicio 10

Selecciona el último hijo BBB del elemento AAA

```
<AAA>
        <BBB/>
        <BBB/>
        <BBB/>
        <BBB/>
</AAA>
```

5.7.11. Ejercicio 11

Selecciona todos los atributos 'id'

```
   <AAA>
        <BBB id = "b1"/>
        <BBB id = "b2"/>
        <BBB name = "bbb"/>
        <BBB/>
   </AAA>
```

5.7.12. Ejercicio 12

Selecciona los elementos BBB que contienen un atributo 'id'

```
<AAA>
        <BBB id = "b1"/>
        <BBB id = "b2"/>
        <BBB name = "bbb"/>
        <BBB/>
</AAA>
```

5.7.13. Ejercicio 13

Selecciona los elementos BBB que contienen algún atributo

```
<AAA>
        <BBB id = "b1"/>
        <BBB id = "b2"/>
        <BBB name = "bbb"/>
        <BBB/>
</AAA>
```

5.7.14. Ejercicio 14

Selecciona los elementos BBB cuyo atributo 'id' tiene por valor 'b1'

```
<AAA>
        <BBB id = "b1"/>
        <BBB name = " bbb "/>
        <BBB name = "bbb"/>
</AAA>
```

5.7.15. Ejercicio 1.15

Selecciona todos los elementos cuyo nombre se inicie con la letra B

```
   <AAA>
      <BCC>
         <BBB/>
         <BBB/>
         <BBB/>
      </BCC>
      <DDB>
         <BBB/>
         <BBB/>
      </DDB>
      <BEC>
         <CCC/>
         <DBD/>
      </BEC>
   </AAA>
```

5.7.16. Ejercicio 16

Selecciona los elementos que contienen dos hijos BBB

```
<AAA>
      <CCC>
         <BBB/>
         <BBB/>
         <BBB/>
      </CCC>
      <DDD>
         <BBB/>
         <BBB/>
      </DDD>
      <EEE>
         <CCC/>
         <DDD/>
      </EEE>
   </AAA>
```

5.7.17. Ejercicio 17

Selecciona todos los elementos CCC y BBB

```
   <AAA>
      <BBB/>
      <CCC/>
      <DDD>
         <CCC/>
      </DDD>
      <EEE/>
   </AAA>
```

5.7.18. Ejercicio 18

Selecciona todos los elementos BBB y los elementos EEE que son hijos del elemento raíz AAA

```
   <AAA>
      <BBB/>
      <CCC/>
      <DDD>
         <CCC/>
      </DDD>
      <EEE/>
   </AAA>
```