

# Tema 1: Obtención dinámica de información

Máster Universitario en Ingeniería Informática  
Extracción y Explotación de la Información

# Índice

- Formato de datos XML
  - ¿Qué es XML?
  - Estructura XML (tipos de nodo)
  - Ejemplo
  - Modelo de datos
  - Espacios de nombre (namespaces)
- Lenguajes de búsqueda en XML
- Formato de datos JSON
- Información de redes sociales

# ¿Qué es XML?

La integración de datos en modelos relacionales es bastante conocida, ya que es un modelo sencillo de comprender.

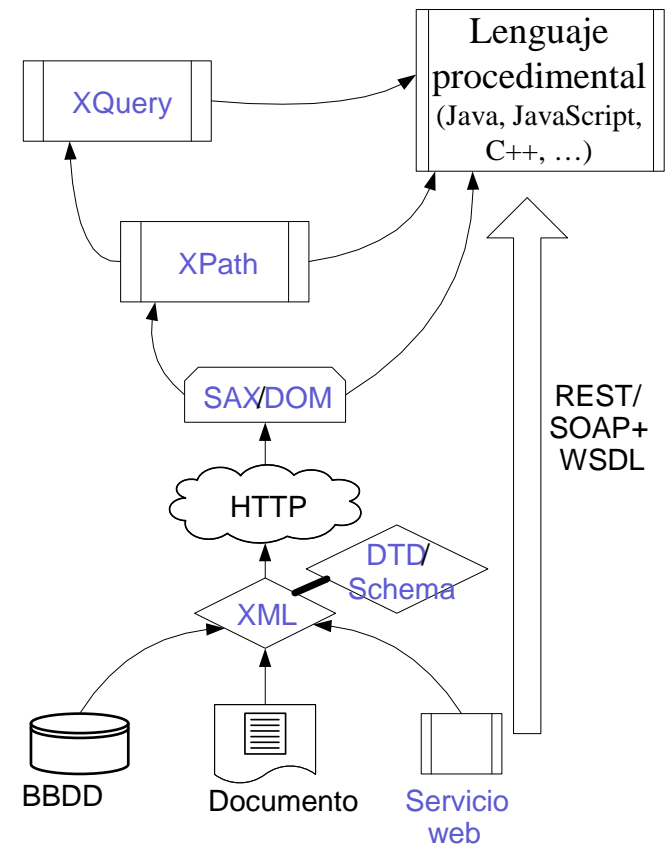
Var A	Var B
$a_1$	$b_1$
$a_2$	$b_2$
...	...

- Sin embargo, los datos no siempre están en formato relacional (Excel, tablas web, ...).
- Se puede emplear un estándar para exportar (e importar) datos.
- Un ejemplo de este estándar sería XML (eXtensible Markup Language).

# ¿Qué es XML?

Se trata de un formato con estructura jerárquica y fácilmente legible.

- Muy relacionado con HTML, siempre analizable (parseable).
- Es un lenguaje oficial de datos: reúne documentos y datos estructurados.
- Combina datos y el esquema de los mismos (estructura).
- Es el núcleo de un entorno más amplio:
  - Datos – XML
  - Esquema – DTD y XML esquema
  - Acceso por programación – DOM y SAX
  - Consulta – XPath, XSLT, XQuery
  - Programas distribuidos– Servicios web



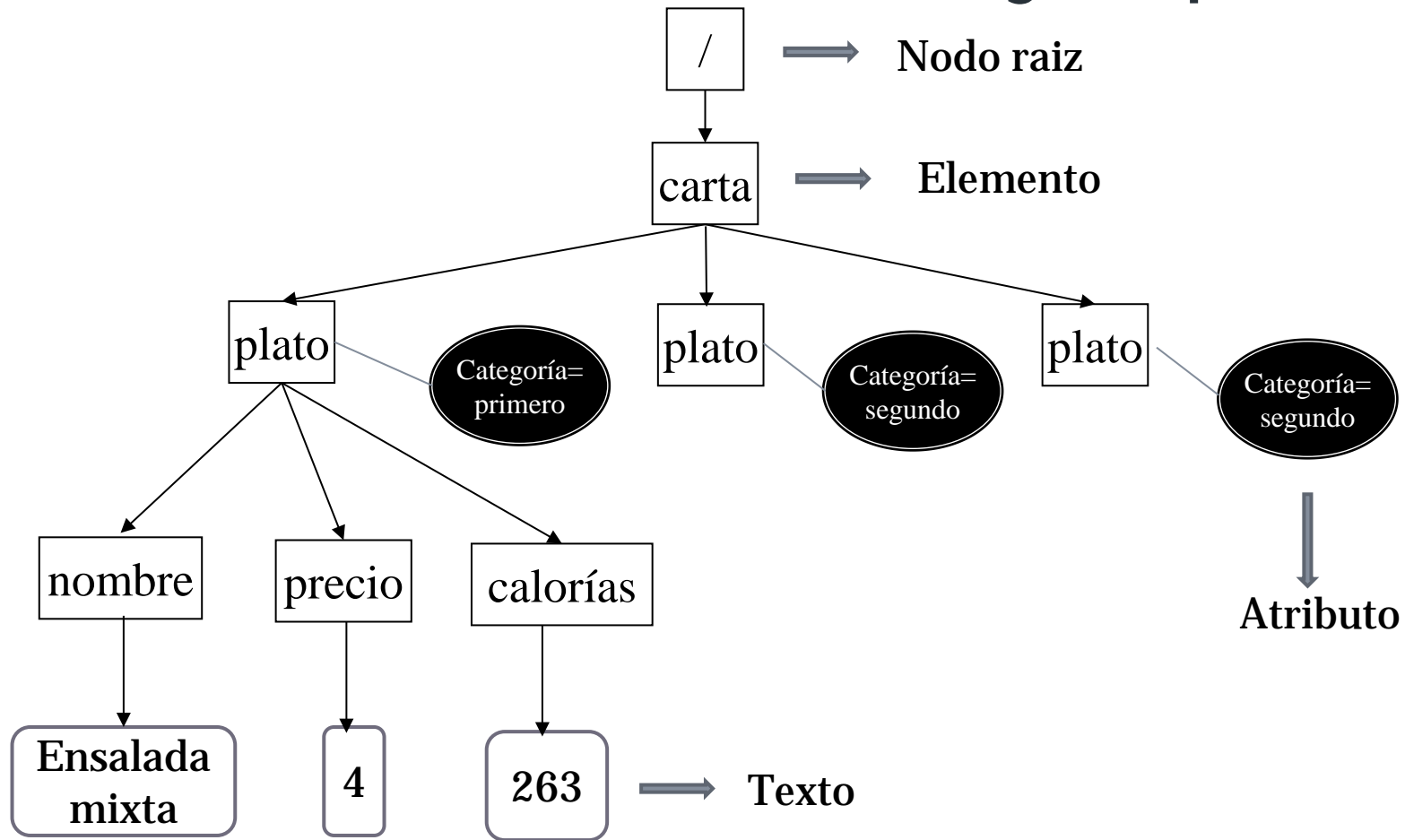
# Estructura XML

- La estructura jerárquica de un documento XML se denomina árbol XML.
- Un árbol XML consiste de varios tipos de nodos organizados en una estructura arborescente.
- Los nodos en un árbol XML pueden ser:
  - (1) nodos de texto que corresponden a un fragmento de información,
  - (2) nodos elementales que definen una agrupación local de información representada por sus descendientes,
  - (3) nodos atributo que completan la información del nombre de un nodo elemental,
  - (4) nodos comentarios,
  - (5) nodos con instrucciones de procesamiento,
  - (6) nodo raíz que representa el documento entero.

# Un documento XML de ejemplo

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<carta>
  <plato categoria="primero">
    <nombre>Ensalada mixta</nombre>
    <precio>4</precio>
    <calorias>263</calorias>
  </plato>
  <plato categoria="segundo">
    <nombre>Paella</nombre>
    <precio>7</precio>
    <calorias>754</calorias>
  </plato>
  <plato categoria="segundo">
    <nombre>Pollo al horno</nombre>
    <precio>5</precio>
    <calorias>488</calorias>
  </plato>
</carta>
```

# Un documento XML de ejemplo



# Modelo de datos XML

- **Elementos**

```
<plato categoria="primero">  
  <nombre>Ensalada mixta</nombre>.....  
</plato>
```

- Estructura jerárquica con parejas de etiquetas (tags) de apertura y cierre <>..  
</>
- Pueden incluir otros elementos anidados
- Pueden incluir atributos dentro de las etiquetas de apertura <>
- Múltiples elementos pueden tener el mismo nombre
- El orden importa

- **Atributos**

```
categoria="primero"
```

- Valores nombrados – no jerárquico
- No puede haber varios atributos con el mismo nombre por elemento.
- El orden no importa



# XML bien formateado: Siempre parseable

Cualquier documento XML debe atender unas condiciones en su estructura

- El comienzo-preámbulo-

`<?xml version="1.0" encoding="utf-8"?>`

- Un único nodo raíz
- Todo los tags `<>` de abrir tienen su correspondiente `</>` de cierre. De forma especial:

`<tag/>` abreviado para tags vacíos (equivalente a `<tag></tag>`)

- Atributos sólo aparecen una vez en un elemento
- XML es sensible mayúsculas.

# Espacios de nombres en XML

- En ocasiones, cuando se quieren combinar documentos XML de diferentes aplicaciones pueden surgir algunos conflictos; p.e., con el nombre de los elementos.

```
<plato categoría="primero">  
  <nombre>Ensalada</nombre>.....  
</plato>
```

```
<alumno centro="ULL">  
  <nombre>Juan García</nombre>.....  
</alumno>
```

- Esto se evita utilizando un prefijo de nombres

```
<m:plato categoría="primero">  
  <m:nombre>Ensalada</m:nombre>.....  
</m:plato>
```

```
<a:alumno centro="ULL">  
  <a:nombre>Juan García</a:nombre>.....  
</a:alumno>
```

# Espacios de nombres en XML

- Al utilizar prefijos en XML se debe definir un espacio de nombres mediante el atributo xmlns (xmlns:prefix="URI")

```
<m:plato xmlns:m="http://www.ull.es/menu" categoría="primero">  
  <m:nombre>Ensalada</m:nombre>.....  
</m:plato>
```

```
<a:alumno xmlns:a="http://www.ull.es/alumno" centro="ULL">  
  <a:nombre>Juan García</a:nombre>.....  
</a:alumno>
```

- En este ejemplo, el atributo xmlns en el tag <name> le da a los prefijos m: y h: un namespace calificado.

# Espacios de nombres en XML

- Los namespaces pueden ser declarados en el elemento raíz:

```
<root xmlns:m="http://www.ull.es/menu" xmlns:a="http://www.ull.es/alumno">
  <m:plato categoría="primero">
    <m:nombre>Ensalada</m:nombre>.....
  </m:plato>

  <a:alumno xmlns:a="http://www.ull.es/alumno" centro="ULL">
    <a:nombre>Juan García</a:nombre>.....
  </a:alumno>
</root>
```

- Obsérvese que el URI (Uniform Resource Identifier) de xmlns es una cadena de caracteres para identificar un recurso en internet, lo más habitual es una URL (Uniform Resource Locator) identificando un dominio.
- El namespace URI no se utiliza por los XML parsers para buscar información. Su propósito es dar un nombre único al namespace.

# Espacios de nombres en XML

- Definir un namespace por defecto nos evita utilizar prefijos en todos los elementos hijos para dicho namespace.

*namespace por defecto  
para nombres no calificados*

*Define calificador "a"*

```
<root xmlns="http://www.w3.org/TR/html4/" xmlns:a="http://www.ull.es/alumno" >
  <plato categoría="primero">
    <nombre>Ensalada</nombre>.....
  </plato>

  <a:alumno centro="ULL">
    <a:nombre>Juan García</a:nombre>.....
  </a:alumno>
</root>
```

# Índice

- ✓ XML data model
- Lenguajes de búsqueda en XML
  - XPath
  - Xquery
  - Actividad: Captura de XML en R
- Formato de datos JSON
- Información de redes sociales

# Lenguajes de búsqueda en XML

- Es preciso definir un lenguaje de búsqueda como parte del procesamiento de datos.
- No están tan desarrollados como el lenguaje relacional aunque se pueden implementar muchas operaciones de consulta.
- Secuencia de desarrollo:
  - XPath
  - XQuery
- Xpath es un lenguaje estándar W3C que permite recorrer el árbol XML desde la raíz como si fuera un grafo dirigido.
  - Se pueden establecer restricciones sobre los valores a devolver en la búsqueda.
  - XPath retorna un conjunto de nodos con los resultados.
  - Xpath es casi un pequeño lenguaje de programación; tiene funciones, tests y expresiones.
- Los ejemplos que se mostrarán en esta sección se pueden ejecutar con el software BaseX, <http://basex.org/> (sudo apt-get install basex)

# XPath

- En su forma más simple, un Xpath parece una ruta de localización (location path) en un sistema de ficheros:

`/path/subpath/.../morepath`

- Una ruta de localización en Xpath describe la ruta desde un punto a otro del árbol XML y retorna el conjunto de nodos que se encuentran al final del path.

`/carta/menu/plato`

Resultado:

```
<plato categoria="primero">
  <nombre>Ensalada mixta</nombre>
  <precio>4</precio>
  <calorias>263</calorias>
</plato>
<plato categoria="segundo">
  <nombre>Paella</nombre>
  <precio>7</precio>
  <calorias>754</calorias>
</plato>
<plato categoria="segundo">
  <nombre>Pollo al horno</nombre>
  <precio>5</precio>
  <calorias>488</calorias>
</plato>
```



# XPath

- Si el path comienza por `/` representa un path **absoluto** desde el nodo raíz. Si comienza por `//` representa un path **relativo** que puede comenzar en cualquier lado del documento.

`/carta/menu/plato`

Resultado:

```
<plato categoria="primero">
<nombre>Ensalada mixta</nombre>
<precio>4</precio>
<calorias>263</calorias>
</plato>
<plato categoria="segundo">
<nombre>Paella</nombre>
<precio>7</precio>
<calorias>754</calorias>
</plato>
<plato categoria="segundo">
<nombre>Pollo al horno</nombre>
<precio>5</precio>
<calorias>488</calorias>
</plato>
```

`//nombre`

Resultado:

```
<nombre>Ensalada mixta</nombre>
<nombre>Paella</nombre>
<nombre>Pollo al horno</nombre>
```

NOTA: This can be expensive, since it involves searching the entire document

# XPath

- Especificando **\*** se pueden especificar todos los elementos que se encuentran en un nivel dado. Observar la diferencia

`/carta/menu/plato`

Resultado:

```
<plato categoria="primero">
  <nombre>Ensalada mixta</nombre>
  <precio>4</precio>
  <calorias>263</calorias>
</plato>
<plato categoria="segundo">
  <nombre>Paella</nombre>
  <precio>7</precio>
  <calorias>754</calorias>
</plato>
<plato categoria="segundo">
  <nombre>Pollo al horno</nombre>
  <precio>5</precio>
  <calorias>488</calorias>
</plato>
```

`/carta/menu/plato/*`

Resultado:

```
<nombre>Ensalada mixta</nombre>
<precio>4</precio>
<calorias>263</calorias>
<nombre>Paella</nombre>
<precio>7</precio>
<calorias>754</calorias>
<nombre>Pollo al horno</nombre>
<precio>5</precio>
<calorias>488</calorias>
```

`//nombre/*`

¿¿??

`/*/*/*calorias`

¿¿??

`//*`

¿¿??

# XPath

- En Xpath existe la noción de nodo contexto (.) y nodo padre (..), de forma análoga a como existe en la especificación de rutas de sistema.

`/carta/menu/plato[2]/nombre/.`

Resultado:

`<nombre>Paella</nombre>`

`/carta/menu/plato[2]/nombre/..`

Resultado:

`<plato>  
<nombre>Paella</nombre>  
<precio>7</precio>  
<calorias>754</calorias>  
</plato>`

- Vemos también como se puede seleccionar un nodo especificando su posición como nodo hijo

`/carta/menu/plato[2]`

`/carta/menu/plato[last()-2]`

`/carta/menu/plato[position()<=2]`

`//nombre[2]    ¿¿??`

NOTA: Counting starts from 1, except in Internet Explorer

# XPath

- También entre corchetes se puede filtrar el conjunto de nodos de acuerdo a una condición sobre el path (predicados)

`/carta/menu/plato[nombre="Tortilla"]`

- Si quisiéramos seleccionar los platos que están calificados como “segundos” resulta preciso hacer referencia al atributo “categoria” de los nodos “platos”. Para ello, se utiliza el nombre del atributo precedido de @

`/carta/menu/plato[@categoria="segundo"]`

- ¿Cómo obtener el valor de un atributo de un nodo?

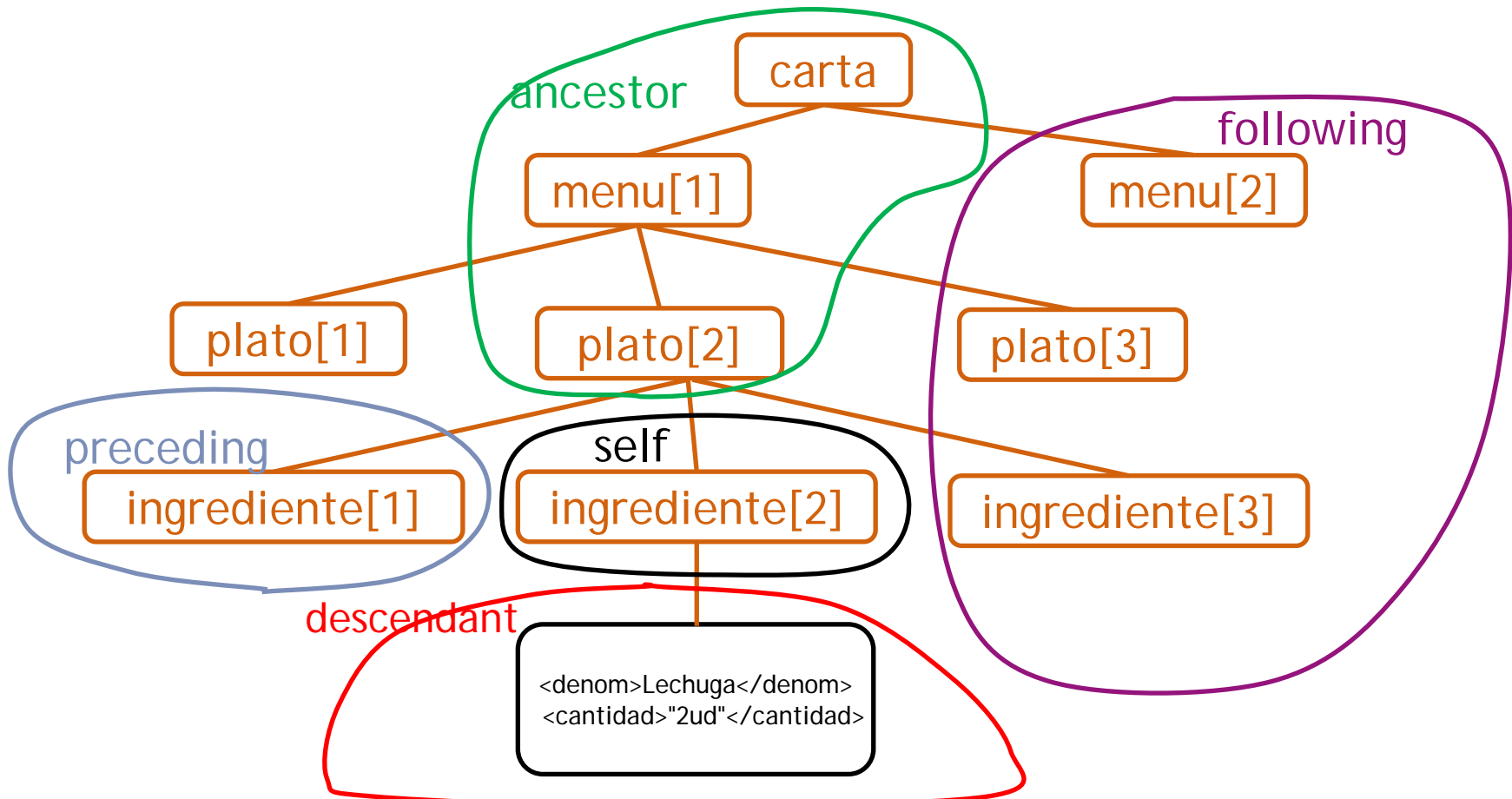
`//menu[1]/plato[2]/@categoria/¿?`

# XPath

- Hasta ahora hemos visto cómo las rutas en Xpath nos permiten descender (o incluso subir) el árbol XML.
- Pero hay estructuras más complejas (ejes) para recorrer en distintos sentidos
  - `self::path-step`
  - `child::path-step`
  - `descendant::path-step`
  - `descendant-or-self::path-step`
  - `preceding-sibling::path-step`
  - `preceding::path-step`
  - `parent::path-step`
  - `ancestor::path-step`
  - `ancestor-or-self::path-step`
  - `following-sibling::path-step`
  - `following::path-step`

# XPath

- De forma ilustrativa



# XPath

- Ejemplos:

`//menu[1]/plato[2]/ingrediente[2]/self::*`

`//menu[1]/plato[2]/ingrediente[2]/preceding::plato`

`//menu[1]/plato[2]/ingrediente[2]/preceding::ingrediente`

`//menu[1]/plato[2]/ingrediente[2]/following::menu`

`//menu[1]/plato[2]/descendant::nombre`

`//menu[1]/plato[2]/child::*`

`//menu[1]/plato[2]/following::*`

`//menu[1]/plato[2]/following-sibling::*`

# XPath

- Existen algunas funciones que pueden ser utilizadas en Xpath para filtrar los tipos de nodos:

`text()`, `number()`, `comment()`, `element()`, `attribute()`

- Se pueden ver varios ejemplos a continuación:

<code>//menu[1]/plato[2]/nombre</code>	<code>//menu[1]/plato[2]/nombre/text()</code>
<code>//menu[1]/plato[2]/calorias</code>	<code>//menu[1]/plato[2]/calorias/number()</code>
<code>//menu[1]/plato[2]/ingrediente</code>	<code>//menu[1]/plato[2]/ingrediente/element()</code>
<code>//menu[1]/plato[2]/attribute()</code>	

- Otras funciones que pueden ser de utilidad son:

<code>count(<i>elem</i>)</code> : cuenta el número de elementos <i>elem</i>	<code>//menu/count(plato)</code>
<code>sum(<i>values</i>)</code> : suma los valores de <i>values</i>	<code>//menu/sum(plato/calorias/number())</code>
<code>name()</code> : devuelve el nombre del elemento	<code>//*[name()='plato']</code>
<code>starts-with(<i>arg1</i>,<i>arg2</i>)</code> : comprueba si el elemento <i>arg1</i> comienza por <i>arg2</i>	<code>//*[starts-with(name(),'ingr')]</code>
<code>contains(<i>arg1</i>,<i>arg2</i>)</code> : comprueba si el elemento <i>arg1</i> contiene <i>arg2</i>	<code>//*[contains(name(),'oria')]</code>



# XPath

- Otras funciones que pueden ser de utilidad son:

`string(elem)`: convierte a string el elemento *elem*      `//menu[1]/plato[2]/string(nombre)`

`number(elem)`: convierte a number el elemento *elem*      `//menu[1]/plato[2]/number(nombre)`

# XPath

- XPath es un lenguaje de anotación para navegar en documentos XML.
- XPath es utilizado para selección de información mediante patrones de búsqueda y puede ser combinado junto con XSLT y Xquery.
- El origen de XPath fué el de un lenguaje simple (1.0) pero su interacción con lenguajes más complejos como Xquery ha hecho que se amplíe (2.0).

# Xquery

- Es un lenguaje de manipulación de XML computacionalmente completo.
- Xquery es para XML lo que SQL para bases de datos.
- Xquery es construido sobre expresiones XPath y representa un estándar W3C.
- Xquery es soportado por la mayor parte de los motores de bases de datos (IBM, Oracle, Microsoft, etc..)
- Permite definir lo que se conoce como expresiones FLWOR (“flower”) que soportan iteraciones y creación de variables.
  - *For*: crea una secuencia de nodos
  - *Let*: asocia una secuencia a una variable
  - *Where*: filtra los nodos por medio de una condición
  - *Order by*: ordena los nodos
  - *Return*: se evalúa 1 vez por cada nodo

# Xquery

- Veamos unas expresiones de ejemplo:
  - **Obtener los platos con precio entre 4 y 5€ y ordenarlos por las calorías**

```
let $menu := doc("menu.xml")
for $plato in $menu//plato
let $nombre:=$plato/nombre
where $plato/precio>=4 and $plato/precio<=5
order by $plato/calorias
return $plato
```

- **Obtener las categorías de platos**

```
let $menu := doc("menu.xml")
for $categ in distinct-values($menu//plato/@categoria)
return <categoria name="{ $categ}" />
```

# Xquery

- Veamos unas expresiones de ejemplo:
  - **Obtener las categorías de platos con la suma total de calorías**

```
let $menu := doc("menu.xml")
for $categ in distinct-values($menu//plato/@categoria)
let $platos:=$menu//plato[@categoria=$categ]
order by $categ
return <categoria name="{ $categ}" totalCal="{sum($platos/calorias)}" />
```

# Xquery

- Veamos unas expresiones de ejemplo con if .. then .. else:
  - **Obtener los platos bajos en calorías y altos en calorías**

```
for $plato in doc("menu.xml")//plato
return
if($plato/calorias<300) then
<plato calorias="baja">{$plato/nombre}</plato>
else <plato calorias="alta">{$plato/nombre}</plato>
```

# Xquery

- Veamos como hacer un join de distintos documentos:

- **Obtener el precio de comandas por mesas**

```
for $mesa in doc("comanda.txt")//mesa
for $plato in doc("menu.xml")//plato[nombre = $mesa/platos/nombre]
return
```

```
<mesa>
{$mesa/num}
<plato>
{$plato/nombre}
{$plato/precio}
</plato>
</mesa>
```

Resultado:

```
<mesa>
  <num>1</num>
  <plato>
    <nombre>Arroz</nombre>
    <precio>3</precio>
  </plato>
</mesa>
<mesa>
  <num>1</num>
  <plato>
    <nombre>Pollo al horno</nombre>
    <precio>5</precio>
  </plato>
</mesa>
.....
```

# Xquery

- Veamos como hacer un join de distintos documentos en el where:
  - **Obtener el precio de comandas agrupado por mesas**

```
for $mesa in doc("comanda.txt")//mesa
return
<mesa>
<num>{$mesa/string(num)}</num>
{for $plato in doc("menu.txt")//plato
where $plato/nombre = $mesa/platos/nombre
return <plato>
{$plato/nombre}
{$plato/precio}
</plato>
}
</mesa>
```

Resultado:

```
<mesa>
  <num>1</num>
  <plato>
    <nombre>Arroz</nombre>
    <precio>3</precio>
  </plato>
  <plato>
    <nombre>Pollo al horno</nombre>
    <precio>5</precio>
  </plato>
</mesa>
.....
```



# Xquery

- Observar las diferencias en el uso de `let` y `for`

```
let $x := (1, 2, 3)      Resultado: 1 2 3 a 1 2 3 b
for $y in ("a", "b")
return($x, $y)
```

```
for $x in (1, 2, 3)      Resultado: 1 a 1 b 2 a 2 b 3 a 3 b
for $y in ("a", "b")
return($x, $y)
```

```
let $x := (1, 2, 3)      Resultado: 1 2 3 a b
let $y := ("a", "b")
return($x, $y)
```

# Actividad: Captura de XML en R

- En R existen algunas librerías que permiten el procesamiento de datos XML (por ejemplo, la librería XML)

```
library(XML)
```

```
#1ra forma
```

```
data<-ldply(xmlToList("menu.xml"), data.frame)
```

```
#2nda forma
```

```
doc = xmlParse("menu.xml")
```

```
data <- data.frame(  
  nombre=xmlToDataFrame(getNodeSet(doc, "//plato/nombre"))$text,  
  precio=xmlToDataFrame(getNodeSet(doc, "//plato/precio"))$text,  
  calorias=xmlToDataFrame(getNodeSet(doc, "//plato/calorias"))$text  
)
```

```
#3ra forma
```

```
platos <- getNodeSet(doc, "//plato")
```

```
data <- data.frame(nombre=unlist(lapply(platos,xpathSApply,"./nombre", xmlValue)),  
  precio=unlist(lapply(platos,xpathSApply,"./precio", xmlValue)),  
  calorias=unlist(lapply(platos,xpathSApply,"./calorias", xmlValue))  
)
```

# Actividad: Captura de XML en R

- También podemos utilizar esta funcionalidad para capturar datos de servicios REST utilizando las APIs correspondientes

```
library(curl)
library("RCurl")
```

```
url<-getURL("http://opendatacanarias.es/datos/dataset/d7bfc168-f410-4cee-9149-3d46c52d4b33/resource/f7d0a1ab-afee-4fe5-a9fd-fc82885f3c2d/download/calendarioacademico20162017.xml")
```

```
doc <- xmlParse(url)
```

# Actividad: Captura de XML en R

- Capturar datos de predicción de meteorológica desde la API de <http://www.aemet.es/es/eltiempo/prediccion/municipios>

<http://api.met.no/weatherapi/locationforecast/1.9/?lat=60.10;lon=9.58;msl=70> (consultar la documentación en <http://api.met.no/weatherapi/locationforecast/1.9/documentation> )

- Capturar datos de alguna fuente de datos de [http://ckan.opendatacanarias.es/dataset?res\\_format=XML&res\\_format\\_limit=0&page=1](http://ckan.opendatacanarias.es/dataset?res_format=XML&res_format_limit=0&page=1)

- Capturar datos de salarios desde la API de <https://data.gov.uk/dataset/senior-salaries2> (consultar otras fuentes en <https://data.gov.uk/> )

# Índice

- ✓ XML data model
- ✓ Lenguajes de búsqueda en XML
- Formato de datos JSON
  - ¿Qué es JSON?
  - ¿Qué NO es JSON?
  - Desventajas de JSON vs XML
  - Estructura de JSON
  - Actividad: Captura de JSON en R
- Información de redes sociales

# ¿Qué es JSON?

- JSON, acrónimo de "JavaScript Object Notation", es un formato ligero para el intercambio de datos (<http://www.json.org/> ).
- JSON es un subconjunto de la notación literal de objetos de Javascript
- La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX.
- Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un analizador semántico de JSON.
- En Javascript, JSON se puede analizar de forma nativa usando el procedimiento `eval()`, lo que ha permitido la aceptación de JSON por parte de la comunidad de desarrolladores Ajax/Javascript.
- Es fácil de comprender, manipular y generar.

# ¿Qué NO es JSON?

- Hay que considerar que:
  - JSON no es un formato de documentos
  - JSON no es un lenguaje de marcado (como markdown)
  - Tampoco es un lenguaje de programación
- Es como XML en el sentido:
  - Formato de texto plano
  - Es “auto-descrito” (fácilmente legible para las personas)
  - Tiene estructura jerárquica (los valores pueden contener listas de objetos o valores)
- Pero NO es como XML en estos otros aspectos:
  - Es más ligero y rápido
  - Utiliza objetos tipados (`{ “type”: “object” }`), mientras que XML son cadenas sin tipado que deben ser parseadas en tiempo de ejecución.
  - Propiedades son inmediatamente accesibles a código Javascript

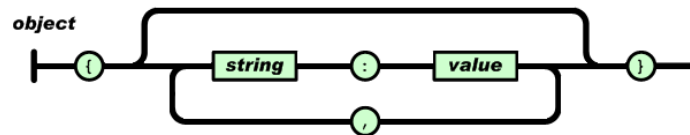
# Desventajas de JSON frente a XML

- Falta de espacios de nombres (namespaces)
- No existen esquemas de validación (XML tiene DTD y plantillas, aunque existe un validador de JSON, JSONlint)
- No es extensible (XML se puede extender añadiendo nuevas etiquetas después de finalizar el diseño y puesto en producción).
- Algunas descripciones y diferencias adicionales se pueden encontrar en <http://www.json.org/xml.html>



# Estructura de JSON

- JSON se compone de 2 estructuras:
  - Los objetos, que son conjuntos desordenados de pares nombre/valor. Un objeto comienza con { y termine con }. Cada nombre es seguido por : y los pares nombre/valor están separados por ,



## XML

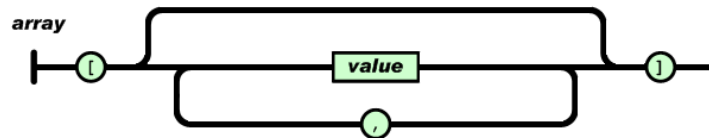
```
<?xml version="1.0" encoding="UTF-8"?>
<menu>
  <plato categoria="primero">
    <nombre>Ensalada</nombre>
    <precio>4</precio>
    <calorias>263</calorias>
  </plato>
  <plato categoria="segundo">
    <nombre>Paella</nombre>
    <precio>7</precio>
    <calorias>754</calorias>
  </plato>
</menu>
```

## JSON

```
{
  "menu": {
    "plato": [
      {
        "nombre": "Ensalada",
        "precio": "4",
        "calorias": "263",
        "categoria": "primero"
      },
      {
        "nombre": "Paella",
        "precio": "7",
        "calorias": "754",
        "categoria": "segundo"
      }
    ]
  }
}
```

# Estructura de JSON

- JSON se compone de 2 estructuras:
  - Y los arrays o listas ordenadas de pares nombre/valor, que comienzan con `[` y termine con `]`. Los pares están separados por `,`



**XML**

```
<?xml version="1.0" encoding="UTF-8"?>
<menu>
  <plato categoria="primero">
    <nombre>Ensalada</nombre>
    <precio>4</precio>
    <calorias>263</calorias>
    <ingredientes name="tomate" qty="0.5">
    <ingredientes name="lechuga" qty="1">
    <ingredientes name="atún" qty="1">
  </plato>
</menu>
```

**JSON**

```
{"menu": {
  "plato": [{"nombre": "Ensalada",
    "precio": "4",
    "calorias": "263",
    "categoria": "primero",
    "ingredientes": [
      {"name": "tomate", "qty": "0.5"},
      {"name": "lechuga", "qty": "1"},
      {"name": "atún", "qty": "1"} ]
    }
  ]
}
```

# Actividad: Captura de JSON en R

- En R también es posible la captura de datos en formato JSON utilizando librerías como la jsonlite

```
library(jsonlite)
doc <- fromJSON("menu.json")
```

- Podemos consultar la API del INE (<http://www.ine.es/dyngs/DataLab/en/manual.html?cid=48>) y capturar las operaciones estadística disponibles.

```
library(jsonlite)
library(curl)
library(RCurl)
```

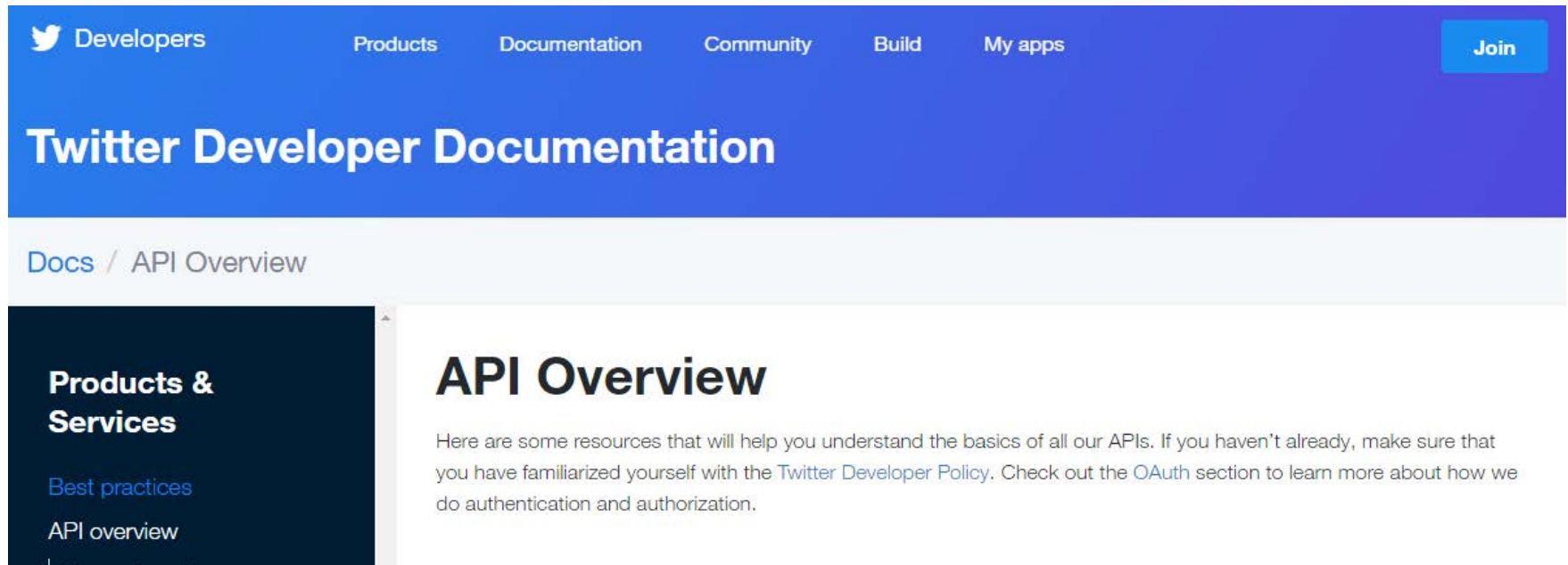
```
tempus.url<- "http://servicios.ine.es/wstempus/js/ES/OPERACIONES_DISPONIBLES"
tempus.oper<- data.frame(fromJSON(txt=as.character(tempus.url)))
```

# Índice

- ✓ XML data model
- ✓ Lenguajes de búsqueda en XML
- ✓ Formato de datos JSON
- Información de redes sociales
  - API de twitter
  - Actividad: Captura de tweets en R

# API de Twitter

- La red social de Twitter dispone de una API para desarrolladores que nos permite obtener información de los tweets de usuarios.



The screenshot shows the Twitter Developer Documentation page. At the top, there is a blue navigation bar with the Twitter logo and the word "Developers" on the left, and links for "Products", "Documentation", "Community", "Build", and "My apps" in the center. A blue "Join" button is on the right. Below the navigation bar, the main heading "Twitter Developer Documentation" is displayed in white. Underneath, a light blue breadcrumb trail reads "Docs / API Overview". On the left side, there is a dark blue sidebar with the text "Products & Services" in white, followed by two links: "Best practices" and "API overview", both in light blue. The main content area on the right has the heading "API Overview" in large, bold black font. Below this heading, a paragraph of text reads: "Here are some resources that will help you understand the basics of all our APIs. If you haven't already, make sure that you have familiarized yourself with the [Twitter Developer Policy](#). Check out the [OAuth](#) section to learn more about how we do authentication and authorization."

Twitter Developers

Products Documentation Community Build My apps

Join

## Twitter Developer Documentation

Docs / API Overview

### Products & Services

[Best practices](#)

[API overview](#)

## API Overview

Here are some resources that will help you understand the basics of all our APIs. If you haven't already, make sure that you have familiarized yourself with the [Twitter Developer Policy](#). Check out the [OAuth](#) section to learn more about how we do authentication and authorization.

# Actividad: Captura de tweets en R

- En R se puede utilizar la librería `twitterR` para este fin:

```
devtools::install_github("twitterR", username="geoffjentry")
library(twitterR)
# Especificar opciones para el acceso
#setup_twitter_oauth(getOption("twitter_consumer_key"),
#                    getOption("twitter_consumer_secret"),
#                    getOption("twitter_access_token"),
#                    getOption("twitter_access_token_secret"))

# Sólo podemos capturar 3200 tweets en el momento, y podemos obtener menos
# dependiendo de la API
emerg_tweets <- userTimeline("112Canarias", n = 3200)
library(purrr)
emerg_tweets_df <- tbl_df(map_df(emerg_tweets, as.data.frame))
```