

Ejercicios resueltos de Lenguajes de Marcas

Los 9 ejercicios de Lenguajes de Marcas de la web, tema a tema: cada uno con su enunciado, los datos que necesitas (código de partida, ejemplos de entrada y salida o la base de datos) y, al final, la solución explicada.

Índice

Fundamentos de Lenguajes de Marcas

1. ¿Están bien anidadas las etiquetas?

3. Resolver los espacios de nombres

Sindicación de contenidos: RSS y Atom

4. Genera un canal RSS 2.0 correcto

Sistemas de Gestión de Información Empresarial

6. Importación incremental con identificadores
externos

Ejercicios largos

7. Resumen de pedidos en XML con ElementTree

9. Observaciones meteorológicas: XPath, XQuery,
XSLT a CSV y DTD

2. Atributos y referencias de entidad

5. Un agregador que mezcla RSS y Atom

8. Catálogo de una editorial: XPath, XQuery, XSLT y
XSD

Fundamentos de Lenguajes de Marcas

1. ¿Están bien anidadas las etiquetas?

Medio · Fundamentos de Lenguajes de Marcas · apuntesdam.com/subject/lenguajes-marcas/topic/fundamentos

Un documento XML bien formado tiene un único elemento raíz y todas sus etiquetas cerradas en orden inverso al de apertura. Lee el documento completo y recorre sus etiquetas (ya están extraídas con una expresión regular; se ignoran `<?...?>`, comentarios y DOCTYPE). Las de cierre son `</nombre>`, las vacías terminan en `/>` y el resto son de apertura (el nombre es lo que va tras `<` hasta un espacio, `/` o `>`). Detén la comprobación en el primer error: «Error en la línea L: `</x>` no tiene etiqueta de apertura», «Error en la línea L: se esperaba `</a>` pero aparece `</b>`», «Error en la línea L: `<x>` es un segundo elemento raíz» (si empieza un elemento cuando la raíz ya se cerró), y al final «Error: falta cerrar `<x>`» (la más interna) o «Error: el documento no tiene elemento raíz». Si todo es correcto: «Bien formado: N elementos, profundidad máxima D, raíz `<r>`» (la raíz tiene profundidad 1). Los nombres distinguen mayúsculas.

Código de partida (python)

```
import re, sys

texto = sys.stdin.read()

def linea(pos):
    return texto.count("\n", 0, pos) + 1

pila = []          # etiquetas abiertas
elementos = 0
profundidad = 0
raiz = None        # nombre de la raíz cuando ya se ha cerrado
for m in re.finditer(r"<[^\>]*>", texto):
    etiqueta = m.group()
    if etiqueta.startswith(("<?", "<!")):
        continue   # declaración, comentarios y DOCTYPE no cuentan
    # TODO: etiquetas de cierre, vacías y de apertura
print("Bien formado")
```

Ejemplo: Catálogo correcto

ENTRADA

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Catálogo de la biblioteca -->
<biblioteca>
  <libro isbn="978-84-1">
    <titulo>El Quijote</titulo>
    <autor>Cervantes</autor>
    <portada archivo="q.jpg"/>
  </libro>
  <libro isbn="978-84-2">
    <titulo>La Regenta</titulo>
  </libro>
</biblioteca>
```

SALIDA ESPERADA

Bien formado: 7 elementos, profundidad máxima 3, raíz `<biblioteca>`

Ejemplo: Cierre cruzado

ENTRADA

```
<pedido>
  <cliente><nombre>Ana</cliente></nombre>
</pedido>
```

SALIDA ESPERADA

Error en la línea 2: se esperaba `</nombre>` pero aparece `</cliente>`

Ejemplo: Dos raíces

ENTRADA

```
<a>
</a>
<b/>
```

SALIDA ESPERADA

Error en la línea 3: es un segundo elemento raíz

Ejemplo: Sin cerrar

ENTRADA

```
<html>
  <body>
    <p>Hola
  </body>
</html>
```

SALIDA ESPERADA

Error en la línea 4: se esperaba </p> pero aparece </body>

2. Atributos y referencias de entidad

Difícil · Fundamentos de Lenguajes de Marcas · apuntesdam.com/subject/lenguajes-marcas/topic/fundamentos

Además de anidar bien, un XML bien formado exige que los atributos tengan valor entre comillas (simples o dobles), sin repetirse en la misma etiqueta y sin «<» en el valor, y que cada «&» empiece una referencia válida: & < > " ';, &#número; o &#xhexadecimal;. Completa las dos funciones. `revisar_etiqueta` recorre los atributos de cada etiqueta de apertura o vacía: el nombre debe empezar por letra o `_` y seguir con letras, dígitos, `_` . : o - (si no, «Línea L, <tag>: nombre de atributo no válido «x»», donde x llega hasta un espacio o =, y se deja de revisar esa etiqueta); sin = → «... el atributo «a» no tiene valor» (y se sigue con el siguiente); sin comillas → «... el valor de «a» no está entre comillas» (se deja la etiqueta); repetido → «... el atributo «a» está repetido»; con < → «... el valor de «a» contiene «<»». Cada atributo con valor cuenta en atributos y su valor pasa por `revisar_referencias`. `revisar_referencias`, para cada &, toma el trozo desde & hasta el primer ; (incluido) sin pasar de un espacio, &, < o ; ; si es válido suma una referencia y si no, «Línea L: referencia no válida «trozo»», con la línea real dentro del texto.

Código de partida (python)

```
import re, sys

texto = sys.stdin.read()
VALIDA = re.compile(r"&(amp|lt|gt|quot|apos|#[0-9]+|#[0-9a-fA-F]+);")

def linea(pos):
    return texto.count("\n", 0, pos) + 1

errores = []
atributos = referencias = 0

def revisar_referencias(cadena, l):
    """Comprueba cada & de la cadena; l es la línea donde se informa."""
    global referencias
    # TODO: referencias de entidad y de carácter

def revisar_etiqueta(etiqueta, l):
    global atributos
    nombre = re.match(r"<\s*([^\s/>]+)", etiqueta).group(1)
    resto = etiqueta[etiqueta.index(nombre) + len(nombre):].rstrip(">").rstrip("/")
    # TODO: atributos de la etiqueta

fin = 0
for m in re.finditer(r"<[>]*>", texto):
    revisar_referencias(texto[fin:m.start()], linea(fin))
    fin = m.end()
    etiqueta = m.group()
    if not etiqueta.startswith(("</", "<?", "<!")):
        revisar_etiqueta(etiqueta, linea(m.start()))
    revisar_referencias(texto[fin:], linea(fin))

if errores:
    print("\n".join(errores))
    print(f"Errores: {len(errores)}")
else:
    print(f"Sin errores: {atributos} atributos y {referencias} referencias correctas")
```

Ejemplo: Tienda con errores

ENTRADA

```
<?xml version="1.0"?>
<tienda nombre="Pan & Café"
web='https://pan.es/?a=1&b=2'>
  <producto id="p1" precio="2,50 &#8364;"
id="p2">Barra & media</producto>
  <producto id=p3 oferta>Croissant &lt;nuevo&gt;
</producto>
  <producto nota="a < b" lid="x">Tarta &copy;
&#x1F370;</producto>
  <producto disponible = "sí" stock="12"/>
</tienda>
```

SALIDA ESPERADA

Línea 3, <producto>: el atributo «id» está repetido
Línea 3: referencia no válida «&»
Línea 4, <producto>: el valor de «id» no está entre comillas
Línea 5, <producto>: el valor de «nota» contiene «<»
Línea 5, <producto>: nombre de atributo no válido «lid»
Línea 5: referencia no válida «©»
Errores: 6

Ejemplo: Documento correcto

ENTRADA

```
<a x="1" y='&quot;dos&quot;';>&lt;3 &#65;&#x42;
</a>
```

SALIDA ESPERADA

Sin errores: 2 atributos y 5 referencias correctas

3. Resolver los espacios de nombres

Muy difícil · Fundamentos de Lenguajes de Marcas · apuntesdam.com/subject/lenguajes-marcas/topic/fundamentos

Un procesador XML no trabaja con prefijos sino con nombres expandidos {URI}local. Lee un documento bien formado y muestra el árbol con el nombre expandido de cada elemento, sangrado dos espacios por nivel; debajo, sangrados dos espacios más, sus atributos «@nombre = valor» (excepto las declaraciones xmlns). Reglas: xmlns="uri" declara el espacio por defecto (xmlns="" lo anula) y xmlns:p="uri" un prefijo; las declaraciones valen para el elemento que las tiene y sus descendientes. Un elemento sin prefijo está en el espacio por defecto; un atributo sin prefijo no está en ningún espacio. El prefijo xml está siempre declarado (<http://www.w3.org/XML/1998/namespace>). Sin espacio de nombres se muestra solo el nombre local. Si un prefijo no está declarado: «Error en la línea L: el prefijo «p» no está declarado» y se termina. Al final, «Espacios de nombres usados: N» y, por orden de primera aparición, « URI: K elemento(s)» contando solo elementos.

Código de partida (python)

```
import re, sys

texto = sys.stdin.read()
XML_NS = "http://www.w3.org/XML/1998/namespace"
ATRIBUTO = re.compile(r'""([\w.-]+\s*=\s*(?:"([^\"]*)"|'\'([^\']*)\')')""')

def linea(pos):
    return texto.count("\n", 0, pos) + 1

ambitos = [{"xml": XML_NS, "": ""}] # pila de ámbitos: prefijo → URI (" es el espacio por defecto)
usos = {} # URI → número de elementos (en orden de primera aparición)
for m in re.finditer(r"<[^>]*>", texto):
    etiqueta = m.group()
    if etiqueta.startswith(("<?", "<!")):
        continue
    if etiqueta.startswith("</"):
        ambitos.pop()
        continue
    nombre = re.match(r"<\s*([^\s/>]+)", etiqueta).group(1)
    atributos = [(a.group(1), a.group(2) if a.group(2) is not None else a.group(3)) for a in
        ATRIBUTO.finditer(etiqueta)]
    actual = dict(ambitos[-1]) # ámbito de este elemento: el del padre más sus declaraciones
    # TODO: declarar, resolver y mostrar el nombre expandido
    if not etiqueta.endswith(">"):
        ambitos.append(actual)
print(f"Espacios de nombres usados: {len(usos)}")
for uri, n in usos.items():
    print(f" {uri}: {n} elemento{'s' if n != 1 else ''}")
```

Ejemplo: Factura con tres vocabularios

ENTRADA

```
<?xml version="1.0"?>
<factura xmlns="urn:facturas"
xmlns:cli="urn:clientes" numero="F-12">
  <cli:cliente cli:nif="12345678Z">
    <cli:nombre xml:lang="es">Ana</cli:nombre>
  </cli:cliente>
  <lineas>
    <linea xmlns="urn:productos"
ref="P1">Teclado</linea>
    <linea xmlns="" ref="P2">Ratón</linea>
  </lineas>
  <total>45.90</total>
</factura>
```

SALIDA ESPERADA

```
{urn:facturas}factura
  @numero = F-12
  {urn:clientes}cliente
    @{urn:clientes}nif = 12345678Z
    {urn:clientes}nombre
      @{http://www.w3.org/XML/1998/namespace}lang
= es
  {urn:facturas}lineas
    {urn:productos}linea
      @ref = P1
    linea
      @ref = P2
  {urn:facturas}total
Espacios de nombres usados: 3
urn:facturas: 3 elementos
urn:clientes: 2 elementos
urn:productos: 1 elemento
```

Ejemplo: Prefijo sin declarar

ENTRADA

```
<doc xmlns:a="urn:a">
  <a:x/>
  <b:y/>
</doc>
```

SALIDA ESPERADA

```
doc
  {urn:a}x
Error en la línea 3: el prefijo «b» no está
declarado
```

Sindicación de contenidos: RSS y Atom

4. Genera un canal RSS 2.0 correcto

Difícil · Sindicación de contenidos: RSS y Atom · apuntesdam.com/subject/lenguajes-marcas/topic/sindicacion-contenidos

Una panadería quiere publicar sus novedades como canal RSS. La primera línea tiene «título|enlace|descripción» del canal y cada una de las siguientes una noticia «fecha ISO 8601|título|enlace|descripción|categoría». Construye el documento con `xml.etree.ElementTree` (que escapa solo los caracteres especiales) y escríbelo: primero la línea `<?xml version="1.0" encoding="UTF-8"?>` y después el `<rss version="2.0">` sangrado con dos espacios (`ET.indent`). El `<channel>` contiene, en este orden, `title`, `link`, `description`, `language` (es-es), `lastBuildDate` (la fecha de la noticia más reciente; se omite si no hay noticias) y los `item` de la más reciente a la más antigua. Cada `item` contiene `title`, `link`, `description`, `category`, `guid` (igual al enlace) y `pubDate`. Las fechas del RSS van en formato RFC 822, como «Mon, 29 Sep 2025 09:00:00 +0200» (`email.utils.format_datetime`).

Código de partida (python)

```
import sys
import xml.etree.ElementTree as ET
from datetime import datetime
from email.utils import format_datetime

lineas = [l for l in sys.stdin.read().splitlines() if l.strip()]
titulo, enlace, descripcion = lineas[0].split("|")
noticias = [] # (fecha, titulo, link, descripcion, categoria)
for l in lineas[1:]:
    fecha, t, link, desc, cat = l.split("|")
    noticias.append((datetime.fromisoformat(fecha), t, link, desc, cat))

# TODO: construye el canal con ElementTree y escríbelo
```

Ejemplo: Panadería

ENTRADA

Panadería La Espiga|https://laespiga.es|Novedades & ofertas del obrador
2025-09-26T18:30:00+02:00|Pan de centeno|https://laespiga.es/centeno|Vuelve el pan de centeno 100 %|Productos
2025-09-29T09:00:00+02:00|Horario de otoño|https://laespiga.es/horario|Abrimos a las 7:30 de lunes a sábado|Avisos
2025-09-28T12:15:00+02:00|Taller: masa madre|https://laespiga.es/taller?dia=4&plazas=10|Aprende a hacer masa madre|Talleres

SALIDA ESPERADA

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
  <channel>
    <title>Panadería La Espiga</title>
    <link>https://laespiga.es</link>
    <description>Novedades & ofertas del obrador</description>
    <language>es-es</language>
    <lastBuildDate>Mon, 29 Sep 2025 09:00:00+0200</lastBuildDate>
    <item>
      <title>Horario de otoño</title>
      <link>https://laespiga.es/horario</link>
      <description>Abrimos a las 7:30 de lunes a sábado</description>
      <category>Avisos</category>
      <guid>https://laespiga.es/horario</guid>
      <pubDate>Mon, 29 Sep 2025 09:00:00+0200</pubDate>
    </item>
    <item>
      <title>Taller: masa madre</title>
      <link>https://laespiga.es/taller?dia=4&plazas=10</link>
      <description>Aprende a hacer <b>masa madre</b></description>
      <category>Talleres</category>
      <guid>https://laespiga.es/taller?dia=4&plazas=10</guid>
      <pubDate>Sun, 28 Sep 2025 12:15:00+0200</pubDate>
    </item>
    <item>
      <title>Pan de centeno</title>
      <link>https://laespiga.es/centeno</link>
      <description>Vuelve el pan de centeno 100 %</description>
      <category>Productos</category>
      <guid>https://laespiga.es/centeno</guid>
      <pubDate>Fri, 26 Sep 2025 18:30:00+0200</pubDate>
    </item>
  </channel>
</rss>
```

5. Un agregador que mezcla RSS y Atom

Muy difícil · Sindicación de contenidos: RSS y Atom · apuntesdam.com/subject/lenguajes-marcas/topic/sindicacion-contenidos

Un agregador junta varios canales en una sola lista. La entrada empieza con «maximo N» y después vienen los documentos separados por líneas «=====»: unos son RSS 2.0 (raíz <rss>) y otros Atom 1.0 (raíz <feed>) en el espacio de nombres <http://www.w3.org/2005/Atom>, que ElementTree escribe como {<http://www.w3.org/2005/Atom>}feed). De cada entrada obtén: la fecha (pubDate en RFC 822 con email.utils.parsedate_to_datetime, o updated en ISO 8601 con datetime.fromisoformat), el nombre del canal (su title), el título y un identificador (guid o, si no tiene, link en RSS; id en Atom). Descarta las repetidas por identificador conservando la primera que aparece, ordena de la más reciente a la más antigua (a igual fecha, por título) y muestra «Canales: C · Entradas: E (duplicadas descartadas: D)» y las N primeras como «AAAA-MM-DD HH:MM UTC · canal · título», con la hora pasada a UTC.

Código de partida (python)

```
import sys
import xml.etree.ElementTree as ET
from datetime import datetime, timezone
from email.utils import parsedate_to_datetime

ATOM = "{http://www.w3.org/2005/Atom}"
partes = sys.stdin.read().split("=====")
maximo = int(partes[0].split()[1]) # «maximo N»
documentos = [ET.fromstring(p.strip()) for p in partes[1:] if p.strip()]

entradas = [] # (fecha UTC, canal, título, identificador)
# TODO: leer RSS y Atom, quitar duplicados, ordenar y mostrar
```

Ejemplo: Dos canales con un duplicado

ENTRADA

```
maximo 10
=====
<rss version="2.0"><channel>
  <title>Apuntes DAM</title>
  <link>https://apuntesdam.com</link>
  <description>Novedades</description>
  <item><title>Nuevo tema de XSLT</title>
  <link>https://apuntesdam.com/xslt</link>
  <guid>tag:apuntesdam,2025:41</guid><pubDate>Mon,
29 Sep 2025 09:00:00 +0200</pubDate></item>
  <item><title>Retos SQL</title>
  <link>https://apuntesdam.com/sql</link>
  <pubDate>Thu, 25 Sep 2025 12:00:00
+0200</pubDate></item>
  <item><title>Ejercicios de Entornos</title>
  <link>https://apuntesdam.com/entornos</link>
  <guid>tag:apuntesdam,2025:40</guid><pubDate>Fri,
26 Sep 2025 18:30:00 +0200</pubDate></item>
</channel></rss>
=====
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Blog de Marta</title>
  <id>urn:uuid:blog-marta</id>
  <updated>2025-09-29T10:00:00Z</updated>
  <entry><title>Mi primer canal Atom</title>
  <id>urn:uuid:m-3</id><updated>2025-09-
29T07:30:00Z</updated><link
href="https://marta.dev/atom"/></entry>
  <entry><title>XPath sin miedo</title>
  <id>urn:uuid:m-2</id><updated>2025-09-
27T16:45:00+02:00</updated><link
href="https://marta.dev/xpath"/></entry>
  <entry><title>Nuevo tema de XSLT (enlace)
</title><id>tag:apuntesdam,2025:41</id>
<updated>2025-09-29T09:05:00+02:00</updated><link
href="https://apuntesdam.com/xslt"/></entry>
</feed>
```

SALIDA ESPERADA

```
Canales: 2 · Entradas: 5 (duplicadas descartadas:
1)
2025-09-29 07:30 UTC · Blog de Marta · Mi primer
canal Atom
2025-09-29 07:00 UTC · Apuntes DAM · Nuevo tema
de XSLT
2025-09-27 14:45 UTC · Blog de Marta · XPath sin
miedo
2025-09-26 16:30 UTC · Apuntes DAM · Ejercicios
de Entornos
2025-09-25 10:00 UTC · Apuntes DAM · Retos SQL
```

6. Importación incremental con identificadores externos

Muy difícil · Sistemas de Gestión de Información Empresarial · apuntesdam.com/subject/lenguajes-marcas/topic/sistemas-gestion-informacion

Los ERP como Odoo importan datos de forma incremental gracias a un identificador externo (external_id): si ya existe, el registro se actualiza; si no, se crea. La entrada tiene una sección [erp] con los clientes actuales «external_id;nif;nombre;email;activo(si/no)» y una sección [importar] con las filas del fichero «external_id;nif;nombre;email». Procesa las filas en orden (la línea N empieza en 1): primero se valida el NIF (8 dígitos y letra, o letra, 7 dígitos y carácter alfanumérico), después el email y después que haya nombre; si falla, «ERROR línea N (id): NIF no válido «x»» (o «email no válido «x»» o «falta el nombre»). Si el NIF ya pertenece a otro registro: «DUPLICADO línea N (id): el NIF x ya es de otro_id». Si el id existe y nada cambia (y está activo): «SIN CAMBIOS id»; si cambia algo: «ACTUALIZAR id: campo antes → después, ...» (en el orden nif, nombre, email), añadiendo «(reactivado)» si estaba inactivo (y «sin cambios de datos» si solo se reactiva). Si no existe: «CREAR id (nombre)». Los cambios se aplican en el momento, así que afectan a las filas siguientes. Termina con «Creados: a · Actualizados: b · Sin cambios: c · Errores: d» y, si hay clientes activos que no aparecen en el fichero, «No incluidos en el fichero: id, id» en el orden del ERP.

Código de partida (python)

```
import re, sys

NIF = re.compile(r"[0-9]{8}[A-Z]|[A-Z][0-9]{7}[A-Z0-9]")
EMAIL = re.compile(r"^\s@+@[^\s@]+\.[^\s@]+")

erp = {} # external_id → {"nif", "nombre", "email", "activo"} (el orden de inserción es el del ERP)
filas = [] # filas del fichero a importar: (external_id, nif, nombre, email)
seccion = None
for linea in sys.stdin.read().splitlines():
    linea = linea.strip()
    if not linea:
        continue
    if linea.startswith("["):
        seccion = linea
        continue
    campos = [c.strip() for c in linea.split(";")]
    if seccion == "[erp]":
        ext, nif, nombre, email, activo = campos
        erp[ext] = {"nif": nif, "nombre": nombre, "email": email, "activo": activo == "si"}
    else:
        filas.append(tuple(campos))

# TODO: procesar la importación y mostrar el informe
```

Ejemplo: Clientes de una gestoría

ENTRADA

```
[erp]
cli_001;12345678Z;Ana García;ana@correo.es;si
cli_002;B1234567X;Talleres Norte
S.L.;admin@tnorte.es;si
cli_003;87654321X;Luis Pérez;luis@correo.es;no
cli_004;A7654321B;Academia
Rioja;info@arioja.es;si
[importar]
cli_001;12345678Z;Ana García;ana@correo.es
cli_002;B1234567X;Talleres Norte
S.L.;facturas@tnorte.es
cli_003;87654321X;Luis Pérez Gil;luis@correo.es
cli_005;C1111111D;Tienda Sur;tienda@sur.es
cli_006;1234567;Pepe Ruiz;pepe@correo.es
cli_007;A7654321B;Otra Academia;otra@academia.es
cli_008;Y7654321G;Marta Díaz;marta.correo.es
cli_005;C1111111D;Tienda Sur S.A.;tienda@sur.es
```

SALIDA ESPERADA

```
SIN CAMBIOS cli_001
ACTUALIZAR cli_002: email admin@tnorte.es →
facturas@tnorte.es
ACTUALIZAR cli_003: nombre Luis Pérez → Luis
Pérez Gil (reactivado)
CREAR cli_005 (Tienda Sur)
ERROR línea 5 (cli_006): NIF no válido «1234567»
DUPLICADO línea 6 (cli_007): el NIF A7654321B ya
es de cli_004
ERROR línea 7 (cli_008): email no válido
«marta.correo.es»
ACTUALIZAR cli_005: nombre Tienda Sur → Tienda
Sur S.A.
Creados: 1 · Actualizados: 3 · Sin cambios: 1 ·
Errores: 3
No incluidos en el fichero: cli_004
```

Ejercicios largos

7. Resumen de pedidos en XML con ElementTree

Difícil · Python · 70 minutos · apuntesdam.com/ejercicios/python/resumen-de-pedidos-xml

La tienda online de una empresa exporta cada noche sus pedidos en XML para el sistema de gestión. El departamento comercial quiere un resumen del día, también en XML, para cargarlo en su cuadro de mando: cuántos pedidos hay en cada estado, cuánto ha comprado cada cliente y cuál es el producto que más se ha vendido.

El fichero lo genera un sistema antiguo y a veces trae pedidos incompletos: sin cliente, con un estado que no existe o con cantidades y precios mal escritos. Esos pedidos se descartan con un aviso que explique el motivo, y el resumen se hace con el resto.

Para leer y escribir XML no se trata como texto con `split`: se usa un analizador. Python trae `xml.etree.ElementTree`, que convierte el documento en un árbol de elementos con sus atributos, y que también sabe construir un árbol nuevo y escribirlo con la sangría correcta y los caracteres especiales escapados.

Requisitos

- La entrada es un documento XML con la raíz `<pedidos fecha="...">` y elementos `<pedido id cliente estado>` que contienen elementos `<línea ref cantidad precio/>`. Si el XML no está bien formado, escribe `XML mal formado (línea N)` con la línea del error; si la raíz no es `pedidos`, El elemento raíz debe ser `<pedidos>`. En los dos casos no se escribe nada más.
- Un pedido se ignora, con el aviso `Aviso: pedido ID ignorado: motivo` (o `?` si no tiene id), cuando: le falta id, cliente o estado (`falta el atributo X`, comprobados en ese orden); su estado no es pendiente, enviado, entregado ni cancelado (`estado desconocido X`); no tiene líneas (`no tiene líneas`); una línea no tiene ref o su cantidad no es un entero mayor que 0 (`línea REF con una cantidad no válida`, con `?` si falta la ref); o un precio no es un número (`línea REF con un precio no válido`).
- Los pedidos válidos cuentan en su estado. Los cancelados no suman importe, ni unidades, ni cuentan como pedidos del cliente. El importe de un pedido es la suma de cantidad × precio de sus líneas, con decimales exactos.
- La salida es un XML `<resumen fecha pedidos importe>` (la fecha de la raíz, el número de pedidos válidos y el importe total con dos decimales y punto), con dos espacios de sangría por nivel, que contiene: `<estados>` con un `<estado nombre pedidos/>` por estado en el orden pendiente, enviado, entregado, cancelado; `<clientes>` con un

<cliente nombre pedidos importe/> por cliente, de mayor a menor importe (a igualdad, por nombre); y
<masVendido ref unidades/> con la referencia de más unidades (a igualdad, la menor alfabéticamente), solo si se ha vendido algo.

- Los avisos se escriben antes del XML.

Formato de la entrada

- Un documento XML de pedidos.

Ejemplo: Los pedidos de un día

ENTRADA

```
<?xml version="1.0" encoding="UTF-8"?>
<pedidos fecha="2026-09-30">
  <pedido id="P-101" cliente="Ana Gil"
    estado="enviado">
    <linea ref="CAB-HDMI" cantidad="2"
      precio="7.95"/>
    <linea ref="RATON-01" cantidad="1"
      precio="24.99"/>
    </pedido>
    <pedido id="P-102" cliente="Muebles Ruiz &
Hijos" estado="entregado">
    <linea ref="SILLA-07" cantidad="4"
      precio="89.90"/>
    </pedido>
    <pedido id="P-103" cliente="Ana Gil"
      estado="cancelado">
    <linea ref="SILLA-07" cantidad="10"
      precio="89.90"/>
    </pedido>
    <pedido id="P-104" cliente="Luis Mora"
      estado="enviado">
    <linea ref="CAB-HDMI" cantidad="3"
      precio="7.95"/>
    <linea ref="TECLADO" cantidad="1"
      precio="abc"/>
    </pedido>
    <pedido id="P-105" cliente="Ana Gil"
      estado="pendiente">
    <linea ref="CAB-HDMI" cantidad="3"
      precio="7.95"/>
    </pedido>
  </pedidos>
```

SALIDA ESPERADA

```
Aviso: pedido P-104 ignorado: línea TECLADO con
un precio no válido
<resumen fecha="2026-09-30" pedidos="4"
importe="424.34">
  <estados>
    <estado nombre="pendiente" pedidos="1" />
    <estado nombre="enviado" pedidos="1" />
    <estado nombre="entregado" pedidos="1" />
    <estado nombre="cancelado" pedidos="1" />
  </estados>
  <clientes>
    <cliente nombre="Muebles Ruiz & Hijos"
      pedidos="1" importe="359.60" />
    <cliente nombre="Ana Gil" pedidos="2"
      importe="64.74" />
  </clientes>
  <masVendido ref="CAB-HDMI" unidades="5" />
</resumen>
```

Ejemplo: Pedidos con errores

ENTRADA

```
<pedidos fecha="2026-10-01">
  <pedido cliente="Sin id" estado="enviado">
  <linea ref="A" cantidad="1" precio="1"/></pedido>
  <pedido id="P-2" estado="enviado"><linea
ref="A" cantidad="1" precio="1"/></pedido>
  <pedido id="P-3" cliente="Eva"
    estado="perdido"><linea ref="A" cantidad="1"
      precio="1"/></pedido>
  <pedido id="P-4" cliente="Eva"
    estado="enviado"/>
  <pedido id="P-5" cliente="Eva"
    estado="enviado"><linea cantidad="1" precio="1"/>
  </pedido>
  <pedido id="P-6" cliente="Eva"
    estado="cancelado"><linea ref="B" cantidad="2"
      precio="5"/></pedido>
</pedidos>
```

SALIDA ESPERADA

```
Aviso: pedido ? ignorado: falta el atributo id
Aviso: pedido P-2 ignorado: falta el atributo
cliente
Aviso: pedido P-3 ignorado: estado desconocido
perdido
Aviso: pedido P-4 ignorado: no tiene líneas
Aviso: pedido P-5 ignorado: línea ? con una
cantidad no válida
<resumen fecha="2026-10-01" pedidos="1"
importe="0.00">
  <estados>
    <estado nombre="pendiente" pedidos="0" />
    <estado nombre="enviado" pedidos="0" />
    <estado nombre="entregado" pedidos="0" />
    <estado nombre="cancelado" pedidos="1" />
  </estados>
  <clientes />
</resumen>
```

Código de partida (python)

```
import sys
import xml.etree.ElementTree as ET
from decimal import Decimal, InvalidOperation

ESTADOS = ("pendiente", "enviado", "entregado", "cancelado")

def leer_pedido(nodo):
    """Comprueba un <pedido> y devuelve su id, cliente, estado e importe; lanza ValueError con el
    motivo."""
    # TODO
    raise ValueError("sin hacer")

def main():
    # TODO: analiza el XML, valida cada pedido y escribe el resumen como XML

main()
```

8. Catálogo de una editorial: XPath, XQuery, XSLT y XSD

Difícil · XML · 90 minutos · apuntesdam.com/ejercicios/xml/catalogo-de-una-editorial-xml

Una editorial pequeña publica su catálogo en XML para que las librerías lo carguen en sus programas. El documento tiene un elemento `libro` por título, con el código, el género y los ejemplares en stock como atributos, y el título, los autores (uno o varios), el año, el precio y las páginas como elementos.

Con ese único documento vas a hacer lo que se hace en el día a día con XML: buscar datos con XPath, sacar informes y construir XML nuevo con XQuery, presentarlo como página web con XSLT y, por último, escribir el XML Schema que garantiza que el documento que reciben las librerías es correcto.

Los apartados se pueden hacer en cualquier orden, pero conviene seguirlo: cada herramienta se apoya en la anterior (XQuery y XSLT usan expresiones XPath, y el XSD formaliza lo que has aprendido de la estructura).

Requisitos

- Resuelve cada apartado con la herramienta que indica, sobre el documento `catalogo.xml`.
- Cada apartado se corrige por separado. Las consultas XPath y XQuery se comparan con el resultado de la solución sobre el documento (importan los valores y su orden); las transformaciones XSLT, con la salida esperada (sin contar espacios y saltos de línea); y los esquemas, con documentos que deben aceptar y otros que deben rechazar.
- Cuando un apartado pide «el texto» de algo, la consulta debe devolver cadenas (con `text()` o `string()`), no elementos.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo editorial="Ediciones del Sur" actualizado="2026-09-30">
  <libro codigo="L001" genero="novela" stock="12">
    <titulo>La casa de las palabras</titulo>
    <autor>Marta Ruiz</autor>
    <anio>2019</anio>
    <precio>18.90</precio>
    <paginas>320</paginas>
  </libro>
  <libro codigo="L002" genero="poesia" stock="0">
    <titulo>Mar adentro, versos</titulo>
    <autor>Julio Peña</autor>
    <anio>2021</anio>
    <precio>12.50</precio>
    <paginas>96</paginas>
  </libro>
  <libro codigo="L003" genero="ensayo" stock="5">
    <titulo>Breve historia de la imprenta</titulo>
    <autor>Ana Gil</autor>
    <autor>Luis Mora</autor>
    <anio>2017</anio>
    <precio>24.00</precio>
    <paginas>410</paginas>
  </libro>
  <libro codigo="L004" genero="novela" stock="3">
    <titulo>El último faro</titulo>
    <autor>Marta Ruiz</autor>
    <anio>2023</anio>
    <precio>21.50</precio>
    <paginas>288</paginas>
  </libro>
  <libro codigo="L005" genero="infantil" stock="25">
    <titulo>Lola y el dragón azul</titulo>
    <autor>Pablo Sanz</autor>
    <anio>2022</anio>
    <precio>9.95</precio>
    <paginas>48</paginas>
  </libro>
  <libro codigo="L006" genero="novela" stock="0">
    <titulo>Ciudades de niebla</titulo>
    <autor>Elena Ríos</autor>
    <anio>2015</anio>
    <precio>19.90</precio>
    <paginas>352</paginas>
  </libro>
  <libro codigo="L007" genero="ensayo" stock="8">
    <titulo>Pensar con datos</titulo>
    <autor>Luis Mora</autor>
    <anio>2024</anio>
    <precio>27.50</precio>
    <paginas>264</paginas>
  </libro>
  <libro codigo="L008" genero="infantil" stock="14">
    <titulo>El bosque que cantaba</titulo>
    <autor>Sara Molina</autor>
    <autor>Pablo Sanz</autor>
    <anio>2020</anio>
    <precio>11.00</precio>
    <paginas>40</paginas>
  </libro>
</catalogo>
```

Apartados

- El texto de los títulos de las novelas, en el orden del documento.
- El texto del título del libro más caro.
- Cuántos libros no tienen ningún ejemplar en stock.
- El texto de los títulos de los libros escritos por más de un autor, en el orden del documento.

- Un informe por género, en orden alfabético: una cadena por género con el formato `género (N): precio medio X €`, donde N es el número de libros y X el precio medio redondeado a dos decimales (sin ceros sobrantes: `25.75`, `20.1`).
- La oferta del mes: para cada libro con más de 10 ejemplares, construye `<oferta codigo="..."><titulo>...</titulo><precio>...</precio></oferta>` con su código, su título y su precio rebajado un 20 % redondeado a dos decimales, ordenadas de menor a mayor precio rebajado.
- Transforma el catálogo en una tabla HTML: `<table>` con la fila de cabecera `<tr><th>Título</th><th>Autores</th><th>Precio</th></tr>` y una fila por libro, ordenadas por título, con los autores separados por coma y espacio, el precio con dos decimales seguido de un espacio y € (`18.90 €`), y el atributo `class="agotado"` en las filas de los libros sin stock.
- Escribe el XML Schema del catálogo: `catalogo` con los atributos obligatorios `editorial` (texto) y `actualizado` (fecha) y uno o más `libro`; cada libro con los atributos obligatorios `codigo` (una L y tres cifras), `genero` (novela, poesía, ensayo o infantil) y `stock` (entero no negativo), y los elementos, en este orden, `titulo`, uno o más `autor`, `anio` (entero de 1450 a 2026), `precio` (decimal no negativo con dos decimales como mucho) y `paginas` (entero positivo).

9. Observaciones meteorológicas: XPath, XQuery, XSLT a CSV y DTD

Difícil · XML · 80 minutos · apuntesdam.com/ejercicios/xml/observaciones-meteorologicas-xml

Varios institutos tienen una estación meteorológica en el tejado y comparten sus datos en un documento XML común: una `estacion` por instituto, con sus atributos (identificador, nombre, provincia y altitud), y dentro un `dia` por cada día con la temperatura máxima, la mínima y la lluvia en milímetros.

El documento declara su DTD en el `DOCTYPE`, pero esa DTD todavía no existe: el último apartado es escribirla. Antes, vas a responder con XPath y XQuery a las preguntas que se hace cualquiera con estos datos (dónde ha llovido más, qué estación es más fría) y a exportarlos en CSV con XSLT para abrirlos en una hoja de cálculo.

Requisitos

- Resuelve cada apartado con la herramienta que indica, sobre el documento `observaciones.xml`.
- Cada apartado se corrige por separado. Las consultas XPath y XQuery se comparan con el resultado de la solución sobre el documento (importan los valores y su orden); las transformaciones XSLT, con la salida esperada (sin contar espacios y saltos de línea); y los esquemas, con documentos que deben aceptar y otros que deben rechazar.
- Cuando un apartado pide «el texto» de algo, la consulta debe devolver cadenas (con `text()` o `string()`), no elementos.

observaciones.xml (xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE observaciones SYSTEM "observaciones.dtd">
<observaciones red="Estaciones de los institutos" periodo="2026-09">
  <estacion id="E01" nombre="Madrid Retiro" provincia="Madrid" altitud="667">
    <dia fecha="2026-09-01"><tmax>31.2</tmax><tmin>18.4</tmin><lluvia>0</lluvia></dia>
    <dia fecha="2026-09-02"><tmax>29.8</tmax><tmin>17.9</tmin><lluvia>0</lluvia></dia>
    <dia fecha="2026-09-03"><tmax>24.1</tmax><tmin>15.2</tmin><lluvia>12.4</lluvia></dia>
    <dia fecha="2026-09-04"><tmax>26.5</tmax><tmin>14.8</tmin><lluvia>1.2</lluvia></dia>
  </estacion>
  <estacion id="E02" nombre="Navacerrada" provincia="Madrid" altitud="1894">
    <dia fecha="2026-09-01"><tmax>19.6</tmax><tmin>9.1</tmin><lluvia>0</lluvia></dia>
    <dia fecha="2026-09-02"><tmax>18.2</tmax><tmin>8.4</tmin><lluvia>3.5</lluvia></dia>
    <dia fecha="2026-09-03"><tmax>12.7</tmax><tmin>5.9</tmin><lluvia>28.1</lluvia></dia>
    <dia fecha="2026-09-04"><tmax>15.0</tmax><tmin>6.3</tmin><lluvia>6.0</lluvia></dia>
  </estacion>
  <estacion id="E03" nombre="Valencia Puerto" provincia="Valencia" altitud="5">
    <dia fecha="2026-09-01"><tmax>30.1</tmax><tmin>23.5</tmin><lluvia>0</lluvia></dia>
    <dia fecha="2026-09-02"><tmax>30.6</tmax><tmin>24.0</tmin><lluvia>0</lluvia></dia>
    <dia fecha="2026-09-03"><tmax>28.4</tmax><tmin>22.1</tmin><lluvia>0.4</lluvia></dia>
    <dia fecha="2026-09-04"><tmax>27.9</tmax><tmin>21.7</tmin><lluvia>15.8</lluvia></dia>
  </estacion>
</observaciones>
```

Apartados

- El nombre (el valor del atributo, como texto) de las estaciones situadas a más de 500 metros de altitud, en el orden del documento.
- La temperatura máxima más alta registrada en todo el periodo (un número).
- Las fechas (sin repetir y en el orden del documento) en las que alguna estación registró más de 10 mm de lluvia.
- Para cada estación, en el orden del documento, una cadena `nombre: media de máximas X °C, lluvia total Y mm`, con la media redondeada a un decimal.
- El día de mayor amplitud térmica (máxima menos mínima) de toda la red, como una única cadena `estación (fecha): amplitud X °C` con la amplitud redondeada a un decimal.
- Exporta a CSV (salida de texto) los días con lluvia: una primera línea `estacion;fecha;tmax;tmin;lluvia` y una línea por cada día con más de 0 mm, con el nombre de la estación, ordenadas por nombre de estación y después por fecha. Cada línea termina con un salto de línea.
- Escribe la DTD `observaciones.dtd`: `observaciones` con uno o más `estacion` y los atributos obligatorios `red` y `periodo`; `estacion` con uno o más `día` y los atributos obligatorios `id` (de tipo ID, único en el documento), `nombre`, `provincia` y `altitud`; `día` con `tmax`, `tmin` y `lluvia` en ese orden y el atributo obligatorio `fecha`; y `tmax`, `tmin` y `lluvia` con texto.

Soluciones

1. ¿Están bien anidadas las etiquetas?

Medio · Fundamentos de Lenguajes de Marcas · apuntesdam.com/subject/lenguajes-marcas/topic/fundamentos

Solución (python)

```
import re, sys

texto = sys.stdin.read()

def linea(pos):
    return texto.count("\n", 0, pos) + 1

pila = []          # etiquetas abiertas
elementos = 0
profundidad = 0
raiz = None        # nombre de la raíz cuando ya se ha cerrado
for m in re.finditer(r"<[>]*>", texto):
    etiqueta = m.group()
    if etiqueta.startswith(("<?", "<!")):
        continue    # declaración, comentarios y DOCTYPE no cuentan
    l = linea(m.start())
    nombre = re.match(r"</?\s*([^\s/>]+)", etiqueta).group(1)
    if etiqueta.startswith("</"):
        # cierre
        if not pila:
            print(f"Error en la línea {l}: </{nombre}> no tiene etiqueta de apertura")
            sys.exit()
        if pila[-1] != nombre:
            print(f"Error en la línea {l}: se esperaba </{pila[-1]}> pero aparece </{nombre}>")
            sys.exit()
        pila.pop()
        if not pila:
            raiz = nombre
        continue
    if raiz is not None and not pila:
        # apertura tras cerrar la raíz
        print(f"Error en la línea {l}: <{nombre}> es un segundo elemento raíz")
        sys.exit()
    elementos += 1
    profundidad = max(profundidad, len(pila) + 1)
    if etiqueta.endswith("/>"):
        # vacía: se abre y se cierra a la vez
        if not pila:
            raiz = nombre
    else:
        pila.append(nombre)

if pila:
    print(f"Error: falta cerrar <{pila[-1]}>")
elif raiz is None:
    print("Error: el documento no tiene elemento raíz")
else:
    print(f"Bien formado: {elementos} elementos, profundidad máxima {profundidad}, raíz <{raiz}>")
```

La estructura de datos natural para comprobar el anidamiento es una pila: cada etiqueta de apertura se apila y cada cierre debe coincidir con la cima, que es la última que se abrió. Si no coincide, las etiquetas se cruzan; si la pila está vacía, sobra el cierre; y si al final queda algo, falta cerrarlo.

La profundidad de un elemento es el tamaño de la pila al abrirlo más uno. Para detectar una segunda raíz basta con recordar el nombre de la raíz cuando la pila vuelve a quedar vacía: cualquier apertura posterior con la pila vacía es otro elemento raíz.

2. Atributos y referencias de entidad

Difícil · Fundamentos de Lenguajes de Marcas · apuntesdam.com/subject/lenguajes-marcas/topic/fundamentos

Solución (python)

```
import re, sys

texto = sys.stdin.read()
VALIDA = re.compile(r"&(amp|lt|gt|quot|apos|#[0-9]+|#[0-9a-fA-F]+);")

def linea(pos):
    return texto.count("\n", 0, pos) + 1

errores = []
atributos = referencias = 0

def revisar_referencias(cadena, l):
    """Comprueba cada & de la cadena; l es la línea donde se informa."""
    global referencias
    for i, c in enumerate(cadena):
        if c != "&":
            continue
        j = i + 1
        while j < len(cadena) and not cadena[j].isspace() and cadena[j] not in "<>":
            j += 1
        trozo = cadena[i:j + 1] if j < len(cadena) and cadena[j] == ";" else cadena[i:j]
        if VALIDA.fullmatch(trozo):
            referencias += 1
        else:
            errores.append(f"Línea {l + cadena.count(chr(10), 0, i)}: referencia no válida «{trozo}»")

def revisar_etiqueta(etiqueta, l):
    global atributos
    nombre = re.match(r"<\s*([^\s/>]+)", etiqueta).group(1)
    resto = etiqueta[etiqueta.index(nombre) + len(nombre):].rstrip(">").rstrip("/")
    lugar = f"Línea {l}, <{nombre}>"
    vistos = set()
    i = 0
    while True:
        while i < len(resto) and resto[i].isspace():
            i += 1
        if i >= len(resto):
            return
        j = i
        while j < len(resto) and not resto[j].isspace() and resto[j] != "=":
            j += 1
        atributo = resto[i:j]
        if not re.fullmatch(r"[A-Za-z_][\w.-]*", atributo):
            errores.append(f"{lugar}: nombre de atributo no válido «{atributo}»")
            return # no se sigue con esta etiqueta
        i = j
        while i < len(resto) and resto[i].isspace():
            i += 1
        if i >= len(resto) or resto[i] != "=":
            errores.append(f"{lugar}: el atributo «{atributo}» no tiene valor")
            continue # se sigue con el siguiente
        i += 1
        while i < len(resto) and resto[i].isspace():
            i += 1
        comilla = resto[i] if i < len(resto) else ""
        cierre = resto.find(comilla, i + 1) if comilla in ("'", '"') else -1
        if cierre < 0:
            errores.append(f"{lugar}: el valor de «{atributo}» no está entre comillas")
            return
        valor = resto[i + 1:cierre]
        i = cierre + 1
        atributos += 1
        if atributo in vistos:
            errores.append(f"{lugar}: el atributo «{atributo}» está repetido")
        vistos.add(atributo)
        if "<" in valor:
```

```

        errores.append(f"{lugar}: el valor de «{atributo}» contiene «<»")
        revisar_referencias(valor, 1)

fin = 0
for m in re.finditer(r"<[^>]*>", texto):
    revisar_referencias(texto[fin:m.start()], linea(fin))
    fin = m.end()
    etiqueta = m.group()
    if not etiqueta.startswith(("</", "<?", "<!")):
        revisar_etiqueta(etiqueta, linea(m.start()))
    revisar_referencias(texto[fin:], linea(fin))

if errores:
    print("\n".join(errores))
    print(f"Errores: {len(errores)}")
else:
    print(f"Sin errores: {atributos} atributos y {referencias} referencias correctas")

```

Los atributos se analizan con un pequeño analizador manual que avanza un índice por el texto de la etiqueta: salta espacios, lee el nombre hasta un espacio o «=», comprueba el «=», y lee el valor hasta la comilla que cierra (la misma que abrió). Cada fallo tiene su mensaje y decide si se puede seguir: sin «=» se continúa con el siguiente atributo; sin comillas o con un nombre inválido ya no se sabe dónde empieza el siguiente, así que se abandona la etiqueta.

Para las referencias, cada «&» se extiende hasta el primer «;» sin cruzar un espacio, otro «&» o un «<», y el trozo se valida con la expresión regular de las referencias permitidas. Por eso «Pan & Café» es un error en XML aunque un navegador lo acepte en HTML: el «&» suelto debe escribirse «&».

La línea de cada error de referencia se calcula sumando a la línea donde empieza el trozo de texto los saltos de línea que hay antes del «&».

Las llaves del nombre expandido se escriben dobladas (`{{` y `}}`) dentro de la f-string, porque una llave sola marcaría una expresión.

4. Genera un canal RSS 2.0 correcto

Difícil · Sindicación de contenidos: RSS y Atom · apuntesdam.com/subject/lenguajes-marcas/topic/sindicacion-contenidos

Solución (python)

```
import sys
import xml.etree.ElementTree as ET
from datetime import datetime
from email.utils import format_datetime

lineas = [l for l in sys.stdin.read().splitlines() if l.strip()]
titulo, enlace, descripcion = lineas[0].split("|")
noticias = [] # (fecha, titulo, link, descripcion, categoria)
for l in lineas[1:]:
    fecha, t, link, desc, cat = l.split("|")
    noticias.append((datetime.fromisoformat(fecha), t, link, desc, cat))

def hijos(padre, pares):
    for etiqueta, valor in pares:
        ET.SubElement(padre, etiqueta).text = valor

rss = ET.Element("rss", version="2.0")
canal = ET.SubElement(rss, "channel")
hijos(canal, [("title", titulo), ("link", enlace), ("description", descripcion), ("language", "es-es")])
noticias.sort(key=lambda n: n[0], reverse=True) # de la más reciente a la más antigua
if noticias:
    hijos(canal, [("lastBuildDate", format_datetime(noticias[0][0]))])
for fecha, t, link, desc, cat in noticias:
    item = ET.SubElement(canal, "item")
    hijos(item, [("title", t), ("link", link), ("description", desc), ("category", cat),
                 ("guid", link), ("pubDate", format_datetime(fecha))])

ET.indent(rss, space=" ")
print('<?xml version="1.0" encoding="UTF-8"?>')
print(ET.tostring(rss, encoding="unicode"))
```

Construir el XML con ElementTree en lugar de concatenar texto garantiza que el resultado esté bien formado:

`SubElement` crea cada hijo en su sitio y, al escribirlo, los caracteres especiales se escapan solos (el «&» del enlace pasa a `&` y el `<b>` de la descripción a `<b>`). Si se pegaran a mano, el canal sería XML inválido y los lectores de RSS lo rechazarían.

Las fechas se ordenan como objetos `datetime` (con su zona horaria) y se escriben en el formato RFC 822 que exige RSS 2.0 con `format_datetime`. La función `hijos` evita repetir la misma línea para cada elemento de texto.

5. Un agregador que mezcla RSS y Atom

Muy difícil · Sindicación de contenidos: RSS y Atom · apuntesdam.com/subject/lenguajes-marcas/topic/sindicacion-contenidos

Solución (python)

```
import sys
import xml.etree.ElementTree as ET
from datetime import datetime, timezone
from email.utils import parsedate_to_datetime

ATOM = "{http://www.w3.org/2005/Atom}"
partes = sys.stdin.read().split("=====")
maximo = int(partes[0].split()[1]) # «maximo N»
documentos = [ET.fromstring(p.strip()) for p in partes[1:] if p.strip()]

entradas = [] # (fecha UTC, canal, título, identificador)
for doc in documentos:
    if doc.tag == "rss":
        canal = doc.find("channel")
        nombre = canal.findtext("title")
        for item in canal.findall("item"):
            fecha = parsedate_to_datetime(item.findtext("pubDate"))
            ident = item.findtext("guid") or item.findtext("link")
            entradas.append((fecha, nombre, item.findtext("title"), ident))
    elif doc.tag == ATOM + "feed":
        nombre = doc.findtext(ATOM + "title")
        for entry in doc.findall(ATOM + "entry"):
            fecha = datetime.fromisoformat(entry.findtext(ATOM + "updated").replace("Z", "+00:00"))
            entradas.append((fecha, nombre, entry.findtext(ATOM + "title"), entry.findtext(ATOM + "id")))

unicas, vistos = [], set()
for e in entradas: # se conserva la primera de cada identificador
    if e[3] not in vistos:
        vistos.add(e[3])
        unicas.append(e)
unicas.sort(key=lambda e: (-e[0].timestamp(), e[2]))

print(f"Canales: {len(documentos)} · Entradas: {len(unicas)} (duplicadas descartadas: {len(entradas) - len(unicas)})")
for fecha, canal, titulo, _ in unicas[:maximo]:
    print(f"{fecha.astimezone(timezone.utc):%Y-%m-%d %H:%M} UTC · {canal} · {titulo}")
```

RSS y Atom guardan lo mismo con nombres distintos, así que lo primero es convertir las entradas de los dos formatos en tuplas iguales (fecha, canal, título, identificador). En Atom todos los elementos están en su espacio de nombres, y ElementTree los nombra con la URI entre llaves: por eso se busca `ATOM + "entry"`.

Las fechas llegan en formatos y zonas horarias distintos; convertidas a `datetime` con zona se pueden comparar directamente, y `astimezone(timezone.utc)` las muestra todas en UTC. El orden usa la tupla (–marca de tiempo, título): de más reciente a más antigua y, a igualdad, por título.

El identificador (guid o id) es lo que permite reconocer la misma noticia publicada en dos canales, aunque el título cambie.

6. Importación incremental con identificadores externos

Muy difícil · Sistemas de Gestión de Información Empresarial · apuntesdam.com/subject/lenguajes-

Solución (python)

```

import re, sys

NIF = re.compile(r"[0-9]{8}[A-Z]|[A-Z][0-9]{7}[A-Z0-9]")
EMAIL = re.compile(r"^[^s@]+@[^s@]+\.[^s@]+$")

erp = {} # external_id → {"nif", "nombre", "email", "activo"} (el orden de inserción es el del ERP)
filas = [] # filas del fichero a importar: (external_id, nif, nombre, email)
seccion = None
for linea in sys.stdin.read().splitlines():
    linea = linea.strip()
    if not linea:
        continue
    if linea.startswith("["):
        seccion = linea
        continue
    campos = [c.strip() for c in linea.split(";")]
    if seccion == "[erp]":
        ext, nif, nombre, email, activo = campos
        erp[ext] = {"nif": nif, "nombre": nombre, "email": email, "activo": activo == "si"}
    else:
        filas.append(tuple(campos))

creados = actualizados = sin_cambios = errores = 0
en_fichero = set()
for n, (ext, nif, nombre, email) in enumerate(filas, start=1):
    en_fichero.add(ext)
    error = None
    if not NIF.fullmatch(nif):
        error = f"NIF no válido «{nif}»"
    elif not EMAIL.fullmatch(email):
        error = f"email no válido «{email}»"
    elif not nombre:
        error = "falta el nombre"
    if error:
        print(f"ERROR línea {n} ({ext}): {error}")
        errores += 1
        continue
    otro = next((e for e, r in erp.items() if e != ext and r["nif"] == nif), None)
    if otro:
        print(f"DUPLICADO línea {n} ({ext}): el NIF {nif} ya es de {otro}")
        errores += 1
        continue
    nuevo = {"nif": nif, "nombre": nombre, "email": email}
    if ext in erp:
        actual = erp[ext]
        cambios = [f"{campo} {actual[campo]} → {nuevo[campo]}" for campo in ("nif", "nombre", "email")
                    if actual[campo] != nuevo[campo]]
        reactivado = not actual["activo"]
        if not cambios and not reactivado:
            print(f"SIN CAMBIOS {ext}")
            sin_cambios += 1
            continue
        print(f"ACTUALIZAR {ext}: {' '.join(cambios) or 'sin cambios de datos'}" + (" (reactivado)" if reactivado else ""))
        actual.update(nuevo, activo=True) # se aplica ya: afecta a las filas siguientes
        actualizados += 1
    else:
        erp[ext] = {**nuevo, "activo": True}
        print(f"CREAR {ext} ({nombre})")
        creados += 1

print(f"Creados: {creados} · Actualizados: {actualizados} · Sin cambios: {sin_cambios} · Errores: {errores}")
fuera = [e for e, r in erp.items() if r["activo"] and e not in en_fichero]
if fuera:
    print(f"No incluidos en el fichero: {' '.join(fuera)}")

```

El identificador externo es lo que hace que una importación se pueda repetir sin duplicar datos: si ya existe, la fila actualiza el registro; si no, lo crea. Las comprobaciones van en orden (formato, después duplicados con otro registro) y, en los registros existentes, se compara campo a campo para informar exactamente de qué cambia.

Los cambios se aplican en el momento sobre el diccionario `erp`, así que una fila posterior ya ve el estado actualizado: la segunda fila de `cli_005` encuentra el cliente recién creado y lo actualiza. Al final se avisa de los clientes activos que no venían en el fichero, que podrían ser bajas que alguien debe revisar.

7. Resumen de pedidos en XML con ElementTree

Difícil · Python · 70 minutos · apuntesdam.com/ejercicios/python/resumen-de-pedidos-xml

Solución (python)

```
import sys
import xml.etree.ElementTree as ET
from decimal import Decimal, InvalidOperation

ESTADOS = ("pendiente", "enviado", "entregado", "cancelado")

def leer_pedido(nodo):
    """Comprueba un <pedido> y devuelve su id, cliente, estado e importe; lanza ValueError con el
    motivo."""
    for atributo in ("id", "cliente", "estado"):
        if not nodo.get(atributo):
            raise ValueError(f"falta el atributo {atributo}")
    if nodo.get("estado") not in ESTADOS:
        raise ValueError(f"estado desconocido {nodo.get('estado')}")
    lineas = nodo.findall("linea")
    if not lineas:
        raise ValueError("no tiene líneas")
    importe = Decimal(0)
    unidades = {}
    for linea in lineas:
        ref, cantidad, precio = linea.get("ref"), linea.get("cantidad", ""), linea.get("precio", "")
        if not ref or not cantidad.isdigit() or int(cantidad) == 0:
            raise ValueError(f"línea {ref or '?'} con una cantidad no válida")
        try:
            precio = Decimal(precio)
        except InvalidOperation:
            raise ValueError(f"línea {ref} con un precio no válido")
        importe += int(cantidad) * precio
        unidades[ref] = unidades.get(ref, 0) + int(cantidad)
    return nodo.get("id"), nodo.get("cliente"), nodo.get("estado"), importe, unidades

def main():
    try:
        raiz = ET.fromstring(sys.stdin.read())
    except ET.ParseError as e:
        print(f"XML mal formado (línea {e.position[0]})")
        return
    if raiz.tag != "pedidos":
        print("El elemento raíz debe ser <pedidos>")
        return

    por_cliente = {} # cliente → [pedidos, importe]
    por_estado = {e: 0 for e in ESTADOS}
    ventas = {}
    total = Decimal(0)
    validos = 0
    for nodo in raiz.findall("pedido"):
        try:
            id_, cliente, estado, importe, unidades = leer_pedido(nodo)
        except ValueError as e:
            print(f"Aviso: pedido {nodo.get('id', '?')} ignorado: {e}")
            continue
        validos += 1
        por_estado[estado] += 1
        if estado == "cancelado":
            continue # se cuenta, pero no suma importe ni unidades
        datos = por_cliente.setdefault(cliente, [0, Decimal(0)])
        datos[0] += 1
        datos[1] += importe
        total += importe
        for ref, n in unidades.items():
            ventas[ref] = ventas.get(ref, 0) + n

    # El resumen se construye como un árbol y se serializa con sangría
```

```

resumen = ET.Element("resumen", fecha=raiz.get("fecha", ""), pedidos=str(validos), importe=f"
    {total:.2f}")
estados = ET.SubElement(resumen, "estados")
for estado in ESTADOS:
    ET.SubElement(estados, "estado", nombre=estado, pedidos=str(por_estado[estado]))
clientes = ET.SubElement(resumen, "clientes")
for cliente, (n, importe) in sorted(por_cliente.items(), key=lambda c: (-c[1][1], c[0])):
    ET.SubElement(clientes, "cliente", nombre=cliente, pedidos=str(n), importe=f"{importe:.2f}")
if ventas:
    ref, n = min(ventas.items(), key=lambda v: (-v[1], v[0]))
    ET.SubElement(resumen, "masVendido", ref=ref, unidades=str(n))
ET.indent(resumen, space="  ")
print(ET.tostring(resumen, encoding="unicode"))

```

```
main()
```

Un analizador de XML comprueba que el documento está bien formado antes de que el programa vea nada, y da acceso a los datos por su estructura (elementos y atributos), no por su posición en el texto. Por eso no le afectan los saltos de línea, el orden de los atributos ni las comillas simples o dobles.

La validación vive en `leer_pedido`, que traduce cada problema en una excepción con un mensaje claro. El bucle principal decide qué hacer con un pedido malo (avisar y seguir) sin mezclarse con las reglas.

Generar el XML con `Element` y `SubElement` en lugar de con `print` garantiza que la salida está bien formada: los nombres con `&` o comillas se escapan automáticamente, algo que es muy fácil olvidar al escribir etiquetas a mano.

Los importes se acumulan con `Decimal` y se formatean con dos decimales y punto, el formato que espera cualquier programa que lea el XML (la coma decimal es para las personas, no para los datos).

8. Catálogo de una editorial: XPath, XQuery, XSLT y XSD

Difícil · XML · 90 minutos · apuntesdam.com/ejercicios/xml/catalogo-de-una-editorial-xml

Apartado 1 (xquery)

```
/catalogo/libro[@genero = 'novela']/titulo/text()
```

Apartado 2 (xquery)

```
/catalogo/libro[precio = max(/catalogo/libro/precio)]/titulo/text()
```

Apartado 3 (xquery)

```
count(/catalogo/libro[@stock = 0])
```

Apartado 4 (xquery)

```
/catalogo/libro[count(autor) > 1]/titulo/text()
```

Apartado 5 (xquery)

```

for $g in distinct-values(/catalogo/libro/@genero)
let $libros := /catalogo/libro[@genero = $g]
order by $g
return concat($g, ' (', count($libros), '): precio medio ', round(avg($libros/precio) * 100) div 100, '
    €')

```

Apartado 6 (xquery)

```

for $l in /catalogo/libro[@stock > 10]
let $precio := round($l/precio * 0.8 * 100) div 100
order by $precio
return <oferta codigo="{ $l/@codigo }"><titulo>{string($l/titulo)}</titulo><precio>{$precio}</precio>
    </oferta>

```

Apartado 7 (xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/catalogo">
    <table>
      <tr><th>Titulo</th><th>Autores</th><th>Precio</th></tr>
      <xsl:for-each select="libro">
        <xsl:sort select="titulo"/>
        <tr>
          <xsl:if test="@stock = 0">
            <xsl:attribute name="class">agotado</xsl:attribute>
          </xsl:if>
          <td><xsl:value-of select="titulo"/></td>
          <td>
            <xsl:for-each select="autor">
              <xsl:value-of select="."/>
              <xsl:if test="position() != last()">, </xsl:if>
            </xsl:for-each>
          </td>
          <td><xsl:value-of select="format-number(precio, '0.00')"/> €</td>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

Apartado 8 (xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="catalogo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="libro" type="tipoLibro" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="editorial" type="xs:string" use="required"/>
      <xs:attribute name="actualizado" type="xs:date" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="tipoLibro">
    <xs:sequence>
      <xs:element name="titulo" type="xs:string"/>
      <xs:element name="autor" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="anio">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="1450"/>
            <xs:maxInclusive value="2026"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="precio">
        <xs:simpleType>
          <xs:restriction base="xs:decimal">
            <xs:minInclusive value="0"/>
            <xs:fractionDigits value="2"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="paginas" type="xs:positiveInteger"/>
    </xs:sequence>
    <xs:attribute name="codigo" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:pattern value="L[0-9]{3}"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="genero" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="novela"/>
          <xs:enumeration value="poesia"/>
          <xs:enumeration value="ensayo"/>
          <xs:enumeration value="infantil"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="stock" type="xs:nonNegativeInteger" use="required"/>
  </xs:complexType>
</xs:schema>
```

Las cuatro tecnologías comparten XPath: es el lenguaje para señalar partes de un documento, y aparece dentro de los `for` de XQuery, en los `select` y `test` de XSLT y en las restricciones de identidad de XSD. Dominar rutas y predicados es la mitad del trabajo con XML.

XQuery es a XML lo que SQL a las bases de datos relacionales: el FLWOR recuerda a `SELECT ... FROM ... WHERE ... ORDER BY`, con la ventaja de que el resultado puede ser XML nuevo. Eso lo hace útil para transformar datos entre sistemas.

XSLT resuelve el mismo problema desde otro ángulo: describe cómo debe ser la salida (aquí, HTML) a partir de plantillas. Por eso es habitual en la publicación de documentos y en la integración entre aplicaciones.

El XML Schema es el contrato del documento: con tipos y restricciones, cualquier librería puede validar el catálogo antes de cargarlo y rechazar un precio negativo o un género desconocido sin escribir código. Los documentos inválidos del ejercicio son justo los errores que el esquema tiene que atrapar.

9. Observaciones meteorológicas: XPath, XQuery, XSLT a CSV y DTD

Difícil · XML · 80 minutos · apuntesdam.com/ejercicios/xml/observaciones-meteorologicas-xml

Apartado 1 (xquery)

```
/observaciones/estacion[@altitud > 500]/@nombre/string()
```

Apartado 2 (xquery)

```
max(//tmax)
```

Apartado 3 (xquery)

```
distinct-values(//dia[lluvia > 10]/@fecha)
```

Apartado 4 (xquery)

```
for $e in /observaciones/estacion
return concat($e/@nombre, ': media de máximas ', round(avg($e/dia/tmax) * 10) div 10, ' °C, lluvia total ', sum($e/dia/lluvia), ' mm')
```

Apartado 5 (xquery)

```
(for $d in //dia
let $amplitud := $d/tmax - $d/tmin
order by $amplitud descending
return concat($d/../@nombre, ' (', $d/@fecha, '): amplitud ', round($amplitud * 10) div 10, ' °C'))[1]
```

Apartado 6 (xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="/observaciones">
    <xsl:text>estacion;fecha;tmax;tmin;lluvia&#10;</xsl:text>
    <xsl:for-each select="estacion/dia[lluvia &gt; 0]">
      <xsl:sort select="../@nombre"/>
      <xsl:sort select="@fecha"/>
      <xsl:value-of select="concat(../@nombre, ';', @fecha, ';', tmax, ';', tmin, ';', lluvia)"/>
      <xsl:text>&#10;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Apartado 7 (xml)

```
<!ELEMENT observaciones (estacion+)>
<!ATTLIST observaciones
  red      CDATA #REQUIRED
  periodo  CDATA #REQUIRED>
<!ELEMENT estacion (dia+)>
<!ATTLIST estacion
  id      ID #REQUIRED
  nombre  CDATA #REQUIRED
  provincia CDATA #REQUIRED
  altitud CDATA #REQUIRED>
<!ELEMENT dia (tmax, tmin, lluvia)>
<!ATTLIST dia fecha CDATA #REQUIRED>
<!ELEMENT tmax (#PCDATA)>
<!ELEMENT tmin (#PCDATA)>
<!ELEMENT lluvia (#PCDATA)>
```

Las consultas XPath se apoyan en la conversión automática a número: comparar un atributo con 500 o un elemento con 10 funciona directamente, y las funciones de agregado (`max`, `avg`, `sum`) hacen el resto.

El eje padre (`..`) es lo que distingue a XPath de una búsqueda de texto: desde un día se llega a su estación sin guardar nada aparte. En XQuery, guardar la amplitud en un `let` permite ordenar por ella y usarla en el resultado.

La salida de texto de XSLT convierte XML en cualquier formato plano (CSV, ficheros de configuración, SQL). Ordenar por dos claves con dos `xsl:sort` funciona igual que un `ORDER BY` con dos columnas.

La DTD es más limitada que el XML Schema (no tiene tipos de datos: una altitud negativa sería válida), pero describe la estructura y garantiza identificadores únicos con `ID`. Por eso sigue apareciendo en documentos antiguos y en el `DOCTYPE` de HTML.