LENGUAJE DE MARCAS - XML

XML

Características

- XML (eXtensible Markup Language)
  - eXtensible: podemos crear nuestras marcas
  - Lenguaje: conjunto de reglas que definen una sintaxis y una gramática
  - Marcado: Método para escribir o incrustar medadatos (información que podemos aportar sobre otro conjunto de datos)
- JSON (JavaScript Object Notation) es una alternativa a XML en AJAX
- SVG es un tipo de dialecto de XML
- Es un Metalenguaje (aporta info extra) extensible: Puede ser usado para definir otros lenguajes
- Subconjunto del SGML
- Desarrollado por el W3C
- Es un lenguaje descriptivo o semántico: las marcas detallan que es lo que se está representando (significado de la info.) pero no especifican cómo representarla. Son flexibles y reutilizables.
- Sintatix: **<etiqueta atributo="valor">contenido</etiqueta>**
  - **Elemento** = etiqueta + contenido ⟶ Componente o estructura mediante la que se organiza el contenido o se indica que acciones se desencadenan) ⟶ **simple** o **complejo** (formado por más elementos ⟶ hijos). Puede haber elementos **vacíos** (sin contenido ⟶ <evento /> || <evento> </evento>).
  - Pueden tener **atributos**: Indican propiedades de los elementos y el **valor (obligatorio)** debe ir entre comillas ⟶ <alumno **id="25"**>Pepe</alumno> ⟶ Se utilizan para distinguir entre elementos del mismo nombre, nos aportan más detalle en la información. <tlf tipo="movil"> <tlf tipo="casa">
  - Todo el documento está englobado en un **solo nodo raíz**

Restricciones etiquetas:

- Son case sensitive (MAYÚSCULA ≠ minúsculas)
- Primer carácter: **una letra o barra baja (_)**
- Sólo pueden contener letras (con o sin tilde), números, puntos (.), guiones medios (-) , barra bajas (_) y dos puntos (:) ⟶ su uso se reserva para definir espacios de nombres.
- ≠ espacios

Restricciones atributos:

- Los identificadores de los atributos se forman de la misma manera que los identificadores de las etiquetas.
- El valor de los atributos siempre entre **comillas** dobles o simples (no pueden mezclarse mes='10")
- No más de un atributo igual en el mismo elemento ⟶ <persona id="" id="">
- Van en la **etiqueta de inicio**.
- Un uso excesivo de atributos puede provocar que el documento XML sea menos legible.
- Los atributos no pueden contener otros elementos o atributos.

- Se suelen usar como "metadatos" ⟶ Por ejemplo identificadores ⟶ un ID con un contador de mensajes, es decir, no una parte de los datos. La información que contiene es lo que se denomina metainformación (información sobre la información). ⟶ <nota ID="001">...</nota><nota ID="002">...</nota>

Entidades:

- En XML algunos símbolos están reservados para el lenguaje. Es necesario utilizar una alternativa(como por ejemplo el texto <a> con los <) utilizar la entidad correspondiente como < ⟶ <a> y

  < ⟶ <

  > ⟶ >

  & ⟶ &

  " ⟶ "

  ‘ ⟶ '

Declaración de XML: <?xml version="1.0" encoding="UTF-8"?>

- No es obligatoria, pero es conveniente incluirla al principio del documento.
- Permite indicar la versión y la codificación empleada en el documento.
- Aunque empieza por <?, NO es una instrucción de procesamiento

Instrucciones de procesamiento

- Comienzan con <? y terminan con ?>
- Usos:
  - Definir hojas de estilo: <?xml-stylesheet href="ruta_archivo.css" type="text/css" ?>
  - Especificar la hoja de estilo XSLT para aplicar transformaciones al documento.
  - Declarar espacios de nombres.
  - Definir el esquema que nos permite VALIDAR el documento.

Comentarios: <!-- Comentario válido en XML ⟶ Pueden estar en cualquier posición excepto: antes de la declaración de XML o dentro de una etiqueta.

Documentos bien formados: Aquellos que son sintácticamente correctos según las reglas y solo tiene un único elemento raíz ⟶ si está bien formado se visualiza correctamente en el navegador.

```
<noticias> ⟶ único nodo raíz
    <noticia>
        <lugar>Alcobendas</lugar>
        <evento fecha="01/02/2022" >
            Comienza el tercer cuatrimestre en el
            módulo de Lenguajes de Marcas
        </evento>
    </noticia>
    ...
</noticias>
```

CSS para XML:

- Usando la instrucción <?xml-stylesheet href="ruta_archivo.css" type="text/css" ?>
- Lo más importante es la propiedad **display** con los valores:
  - **none**: permite ocultar elementos.
  - **block**: modo de visualización de los elementos de tipo bloque, <p>, <h1>, etc. El elemento ocupa toda la ventana en horizontal y ocupa el espacio vertical necesario para alojar el contenido del elemento.
  - **inline**: modo de visualización de etiquetas como , , <a>, etc. El elemento sólo ocupa el espacio necesario para alojar el contenido del elemento. Tienen que estar dentro de elementos de bloque.
  - **list-item**: es el modo de visualización de elementos de listas como <li>, <dd>, <dt>. Los elementos que tienen asociada la propiedad display: list-item ⟶ pueden elegir el estilo de la lista (list-style: disc | circle | square | decimal | lower-roman + inside | outside ⟶ la posición del marcador).

```
escritorio {
  display: list-item;
  list-style: disc inside;
}
```

  - **table** (para asignar comportamiento de tabla a los elementos que haya dentro), **table-row** (actúa como un elemento fila) y **table-cell** (se lo colocarías a las etiquetas que deben trabajar como una celda de tabla) se combinan para mostrar elementos en forma de tabla. Los elementos que tienen asociado alguna de estas propiedades pueden usar la propiedad border-collapse (collapse | separate), estilo del borde...

```
<distribuciones>
  <distribucion id="manjaro">
    <nombre>Manjaro Linux</nombre>
    <web>https://manjaro.org/</web>
    <nacimiento>2011</nacimiento>
  </distribucion>
  <distribucion id="mint">
    <nombre>Linux Mint</nombre>
    <web>www.linuxmint.com/</web>
    <nacimiento>2006</nacimiento>
    <derivada>ubuntu</derivada>
  </distribucion>
  <distribucion id="ubuntu">
    <nombre>Ubuntu</nombre>
    <web>www.ubuntu.com/</web>
    <nacimiento>2004</nacimiento>
    <derivada>debian</derivada>
  </distribucion>
</distribuciones>
```

```
distribuciones {
  display: table;
  margin-bottom: 20px;
  margin-top: 0;
  margin-left: auto;
  margin-right: auto;
}

distribucion {
  display: table-row;
}

distribucion nombre, distribucion web,
distribucion nacimiento {
  display: table-cell;
  border-bottom: 2px solid black;
  padding: 5px;
  padding-right: 10px;
}
```

  - **flex** o **inline-flex** para visualizar los elementos en bloque (block), inline, list-item, tabla, flex...
- Se pueden usar pseudoelementos ::before y ::after ⟶ util para añadir texto a los XML que no se pueden modificar

```
lanzamientos::after {
  content: "DISTRIBUCIONES DE SOFTWARE LIBRE"; ⟶ se añadirá este texto antes del elemento lanzamientos
}
```

- Atributos **class** e **id** en XML: Se pueden añadir atributos class e id al XML (<ley class="lavanda">Nunca necesitarás nada que guardes por si lo necesitas) pero para que el documento se visualice correctamente en los navegadores, es necesario incluir el espacio de nombres http://www.w3.org/1999/xhtml ⟶ se incliye en la etiqueta de inicio del

elemento en el que queramos usar ese espacio de nombres ⟶ Podemos incluirlo en el nodo raíz y se aplicará a todos los demás elementos que contiene.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="1_leyes.css"?>
<leyes xmlns="http://www.w3.org/1999/xhtml">
    <titulo>Humor</titulo>
    <murphy>
        <tipo>Leyes de Murphy</tipo>
        <ley class="amarillo">Si algo puede fallar, fallará. </ley>
        <ley class="lavanda">Si algo no puede fallar, lo hará a pesar de todo.</ley>
    </murphy>
</leyes>
```

Espacios de nombres en XML

- Un espacio de nombres XML es una recomendación W3C para proporcionar elementos y atributos con nombre único en un archivo XML. Se utilizan para diferenciar distintos vocabularios dentro de un XML y poder atribuir, a estos vocabularios, los distintos elementos o atributos del archivo. Permite resolver casos en los que más dos o más elementos tienen el mismo nombre pero su significado es diferente (como el elemento "capital" que puede hacer referencia a una ciudad pero a la vez al dinero de una entidad). En este ejemplo, por lo tanto, se crean dos espacios (uno que hace referencia al significado geográfico y otro al sentido financiero) para poder diferenciarlos.
- Un espacio de nombres se declara con el atributo **xmlns**, cuyo valor debe ser un identificador uniforme de recurso (URI). Las URIs no contiene código alguno, simplemente describe el espacio de nombres a lectores humanos. El hecho de usar una URL para identificar un espacio de nombres reduce la posibilidad de que diferentes espacios de nombres usen identificadores iguales. Las URIs no tienen porqué contener nada, su función es ser únicos.
- La declaración suele incluir un **prefijo** corto con el que los elementos y atributos pueden identificarse ⟶ <nombre_elemento xmlns:prefijo="URI_del_espacio_de_nombres"> y se usan ⟶ <prefijo:nombre_etiqueta></prefijo:nombre_etiqueta>

```
<info:venta xmlns:info="empresa:espacios:info" xmlns:prod="empresa:espacios:prod">
        <prod:producto prod:id="p_1004">Mesa de despacho</prod:producto>
        <prod:precio prod:moneda="euro">Mesa de despacho</prod:precio>
        <prod:unidades>200</prod:unidades>
</info:venta>
```

empresa:espacios:info y empresa:espacios:prod ⟶ espacios de nombre

prod e info ⟶ prefijos

- Los espacios de nombres pueden definirse en el elemento raíz (caso anterior) o, directamente, en los elementos que los vayan a utilizar.

```
<e1:ejemplo xmlns:e1="http://www.abrirllave.com/ejemplo1"> ⟶ están definidos en cada elemento
    <e1:carta>
        <e1:palo>Corazones</e1:palo>
        <e1:numero>7</e1:numero>
    </e1:carta>
    <e2:carta xmlns:e2="http://www.abrirllave.com/ejemplo2">
        <e2:carnes>
            <e2:filete_de_ternera precio="12.95"/>
            <e2:solomillo_a_la_pimienta precio="13.60"/>
        </e2:carnes>
    </e2:carta>
</e1:ejemplo>
```

- Un identificador de recursos uniforme o URI identifica un recurso por su nombre, por su ubicación o por ambos. Engloba URL ("Localizador uniforme de recursos") y el URN ("Nombre Uniforme de Recurso"). Suelen seguir el formato: esquema://máquina/directorio/archivo.
- Espacio de nombres por defecto: cuando no se define un prefijo ⟶ se aplica al elemento en el que se ha declarado y sus elementos descendientes, pero no a sus atributos.

```
<libro xmlns="urn:loc.gov:libros" xmlns:isbn="urn:ISBN:0-395">
        <titulo>Mas barato que una Docena</titulo>
        <isbn:numero>123456</isbn:numero>
</libro>
```

urn:loc.gov:libros ⟶ espacio por defecto y pertenecen a él los elementos <libro> y <titulo>

```
<venta xmlns="empresa:espacios:publicidad" xmlns:venta="empresa:espacios:ventas">
        <producto id="p_1004">Mesa de despacho</producto>
        <venta:precio venta:moneda="euro">Mesa de despacho</venta:precio>
        <unidades>200</unidades>
</venta>
```

Los elementos <venta> <producto> y <unidades> ⟶ pertenecen al espacio por defecto "empresa:espacios:publicidad" pero el atributo "id" no pertenece a ningún espacio de nombres (porque el espacio por defecto no se aplica a los atributos) y el atributo "moneda" pertenece a "empresa:espacios:ventas".

Documentos válidos: Aquellos que, además de estar bien formados, cumplen los requisitos de una definición de estructura: Definición de Tipo de Documento (DTD) y Esquemas.

VALIDACIÓN CON DTD

- Una DTD es un documento que define la estructura de un documento XML: los elementos, atributos, entidades, notaciones, etc, que pueden aparecer, el orden y el número de veces que pueden aparecer, cuáles pueden ser hijos de cuáles, etc. El procesador XML utiliza la DTD para verificar si un documento es válido, es decir, si el documento cumple las reglas del DTD.
- La DTD puede incluirse en el propio documento o en un documento externo:

Interna

```
<!DOCTYPE nombre [
 ... declaraciones ...
]>
```

Externo

<!DOCTYPE nombre SYSTEM "ubi/caci/on**.dtd**"> ⟶ el archivo donde se desarrolle la DTD tiene formato .dtd

- La DTD puede incluirse en el propio documento o en un documento externo:

Declaración de elementos: <!ELEMENT nombreElemento (contenido)>

- Por cada elemento se indica un !ELEMENT

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE deportistas [
<!ELEMENT deportistas (deportista*)>
<!ELEMENT deportista (deporte,nombre)>
<!ELEMENT deporte (#PCDATA)>
<!ELEMENT nombre (#PCDATA)>
]>
```

```
<deportistas>
    <deportista>
        <deporte>Atletismo</deporte>
        <nombre>Jesse Owens</nombre>
    </deportista>
    ...
</deportistas>
```

- Contenido:
  - EMPTY: elemento vacío.
  - (#PCDATA): texto
  - ANY: cualquier cosa.
  - , (coma): elementos en el orden indicado.
  - | (o lógico): contiene uno de los dos elementos ⟶ <!ELEMENT albumesMortadelo (album | CD)>
  - ?: cero o una vez
  - *: cero o varias veces
  - +: mínimo 1 vez o más de una
  - (): agrupar expresiones.

Declaración de atributos: <!ATTLIST nombreElemento nombreAtributo tipoAtributo #valorInicialAtributo >

- Por cada atributo se indica un ATTLIST de las siguientes dos maneras:

  <!ATTLIST **nombreElemento** nombreAtributo1 tipoAtributo1 valorInicialAtributo1>

  <!ATTLIST **nombreElemento** nombreAtributo2 tipoAtributo2 valorInicialAtributo2>

  <!ATTLIST **nombreElemento** nombreAtributo1 tipoAtributo1 valorInicialAtributo1 nombreAtributo2 tipoAtributo2 valorInicialAtributo2>

- Tipos de atributos:
  - CDATA: caracteres (sin restricciones).
  - NMTOKEN: letras, dígitos, y los caracteres punto (.) , guion (-), barra baja (_) y dos puntos (:).
  - NMTOKENS: como NMTOKEN pero también espacios en blanco.
  - Valores: valores de una lista. Lista entre paréntesis, con términos separados por una barra vertical (|). Términos entre comillas simples o dobles si contienen espacios en blanco. ⟶ <!ATTLIST album fecha (1969 | 1970 | 1971) #REQUIRED> o <!ATTLIST dichos saludos ("Hola, ¿cómo estás?" | "Adiós, hasta mañana") #REQUIRED>
  - ID: valor no se puede repetir en otros elementos o atributos.

```
<!DOCTYPE futbol [
 <!ELEMENT futbol (jugador*)>
 <!ELEMENT jugador EMPTY>
 <!ATTLIST jugador nombre NMTOKENS #REQUIRED>
 <!ATTLIST jugador codigo ID #REQUIRED>
]>
```

```
<futbol>
   <jugador nombre="Alfredo Di Stéfano" codigo="D1"/>
   <jugador nombre="Pelé" codigo="D2" />
   <jugador nombre="Johan Cruyff" codigo="D4" />
</futbol> ⟶ códigos no se repiten
```

  - IDREF: valor debe coincidir con el valor del atributo ID de otro elemento.
  - IDREFS: valor es una serie de valores separados por espacios que coinciden con el valor del atributo ID de otros elementos.
  - ENTITY: entidad definida en la DTD.
  - ENTITIES: alguna de las entidades de una lista de entidades definida en la DTD.

- NOTATION: notación definida en la DTD.

- Valores iniciales:
  - #REQUIRED: el atributo es obligatorio, aunque no se especifica ningún valor predeterminado.
  - #IMPLIED: el atributo no es obligatorio y no se especifica ningún valor predeterminado.
  - #FIXED valor: el atributo tiene un valor fijo.
  - valor: el atributo tiene un valor predeterminado.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resoluciones [
    <!ELEMENT resoluciones (resolucion*)>
    <!ELEMENT resolucion EMPTY>
    <!ATTLIST resolucion nombre NMTOKENS #REQUIRED>
    <!ATTLIST resolucion alto CDATA #REQUIRED>
    <!ATTLIST resolucion ancho CDATA #REQUIRED>
]>
```

```
<resoluciones>
    <resolucion nombre="VGA" alto="480" ancho="640" />
    <resolucion nombre="XGA" alto="1024" ancho="768" />
    <resolucion nombre="HD 1080" alto="1920"
    ancho="1080" />
</resoluciones>
```

VALIDACIÓN CON ESQUEMAS (XSD)

- Lenguaje también conocido como XSD (XML Schema Definition) que se utilizan para validar documentos XML y que por lo tanto describe la estructura de un documento XML: qué elementos y atributos deben aparecer, número y orden de los elementos hijo, tipos de datos de los elementos y atributos, sus valores por defecto y fijos...
- Soporta tipos de datos por lo tanto es más sencillo.
- Permite:
  - Describir los elementos que puede contener el archivo.
  - Validar los datos.
  - Definir restricciones.
  - Definir patrones y formatos de datos.
  - Convertir datos entre diferentes tipos.
  - Comunicaciones seguras: permite describir cómo son los datos que va a recibir ⟶ Por ejemplo si enviamos a un receptor una fecha como 04-11-2018 puede interpretarse como 4 de noviembre o 11 de abril (según el país). Con un documento XML con <fecha type="date">2018-11-04</fecha>, su esquema deinirá su tipo de datos "date" que exige ajustarse al formato "YYYYMM-DD" (año, mes, día).
- Los esquemas XML son extensibles puesto que están escritos en XML, por lo tanto: se pueden reutilizar esquemas, crear tipos propios de datos a partir de tipos estándar y hacer referencia a múltiples esquemas en el mismo documento.
- Para usar un esquema XSD, hay que **referenciarlo** desde el archivo XML así (no vamos a usar espacios de nombre y esquemas por que no hace falta saber cómo vincular un XML a un XSD si hay espacios de nombre):

```
<?xml version="1.0" encoding="UTF-8"?>
<email xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ⟶ Indica al parser XML que el documento debe ser validado frente a un esquema
xsi:noNamespaceSchemaLocation="email.xsd"> ⟶ Especifica dónde está el esquema (ruta al esquema)
...
</email>
```

Sintaxis

- El elemento **<xs:schema></xs:schema>** engloba todo el XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" ⟶ Espacio de nombres del que provienen los elementos y tipos de datos del esquema
elementFormDefault="qualified"> ⟶ Cualquier elemento usado en el XML declarado en este esquema debe estar en el espacio de nombres por defecto
...
...
</xs:schema>
```

- Para cada **elemento** que aparezca en el XML, tenemos que decidir si es simple o complejo.
  - **Tipo simple**: Elementos que contienen solo texto (no contienen otros elementos ni atributos) ⟶ simpleType

    <xs:element **name="empleado" type="xs:string"** default="yyy" fixed="yyy" > ⟶ **name** es el nombre del elemento y **type** el tipo de datos al que se corresponde. default y fixed son opcionales, default es el valor por defecto del elemento cuando no recibe ningún valor y fixed indica un valor automático que no se puede cambiar.

    Los tipos de datos predefinidos (datos simples) pueden ser: **xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time.**
  - **Tipo complejo**: Elementos vacíos o no que contienen otros elementos y/o atributos. ⟶ complexType

- Los **atributos** se definen siempre como **tipos simples**.

  <xs:attribute **name="xxx" type="yyy"** default="yyy" fixed="yyy" use="required" > ⟶ igual que en los elementos, la única diferente es **use="required" | use="optional"** si es obligatorio o no el atributo

- Para definir si un elemento es simple o complejo, hay dos maneras:
  - Todo junto: declarando el elemento raíz y definiendo los elementos que lo forman.

```
<xs:element name="email"> ⟶ se declara el elemento sin type o simpleType
  <xs:complexType> ⟶ si en xs:element ≠ type ⟶ se tiene que declarar dentro poniendo xs:complexType
    <xs:sequence>
      <xs:element name="de" type="xs:string"/> ⟶ con los tipos predefinidos (xs:string, xs:integer, etc)
      <xs:element name="para" type="xs:string"/>      solo hay que poner lo que son, no hace falta
      <xs:element name="asunto" type="xs:string"/>    definirlo más (si no restricciones)
      <xs:element name="cuerpo" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="color">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="rojo" />
      <xs:enumeration value="verde" />
      <xs:enumeration value="azul" />
      <xs:enumeration value="negro" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- Separando el elemento del type: el atributo type indica el tipo de datos que forma el elemento **(el elemento será vacío)**

```
<xs:element name="email" type="estructuraCorreo" /> ⟶ por un lado está el elemento vacío con el tipo
<xs:complexType name="estructuraCorreo"> ⟶ por otro lado se define el tipo, poniéndole el mismo nombre definido en el elemento
  <xs:sequence>
    <xs:element name="de" type="xs:string"/>
    <xs:element name="para" type="xs:string"/>
    <xs:element name="asunto" type="xs:string"/>
    <xs:element name="cuerpo" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="color" type="tipoColor" />
<xs:simpleType name="tipoColor">
  <xs:restriction base="xs:string">
    <xs:enumeration value="rojo" />
    <xs:enumeration value="verde" />
    <xs:enumeration value="azul" />
    <xs:enumeration value="negro" />
  </xs:restriction>
</xs:simpleType>
```

- Las **restricciones** (o facetas) permiten definir el rango de valores aceptable para elementos o atributos XML. Se define después de declarar si el elemento es complejo o simple ⟶ Restricción para los valores del elemento minutos entre 0 y 59. Pueden ser:
  - **enumeration:** se define una lista de valores válidos.

```
<xs:element name="car">
 <xs:simpleType>
  <xs:restriction base="xs:string"> ⟶ el atributo base indica el tipo de dato a partir del cual se define la restricción
   <xs:enumeration value="Audi"/>
   <xs:enumeration value="Golf"/>
   <xs:enumeration value="BMW"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

  - Restricciones con length:
    - **length:** especifica el número de caracteres o elementos de lista permitidos.
    - **maxLength**: especifica el número máximo de caracteres o elementos de lista permitidos.
    - **minLength**: especifica el número mínimo de caracteres o elementos de lista permitidos.

```
<xs:element name="password">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:length value="8"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

```
<xs:element name="password">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:minLength value="5"/> ⟶ el valor debe tener un
   <xs:maxLength value="8"/> ⟶ mínimo de 5 caracteres y un máximo de 8 caracteres
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

- Restricciones con valores:
  - **maxExclusive**: especifica los límites superiores para los valores numéricos (sin incluir el límite mayor)
  - **minInclusive**: especifica los límites inferiores para los valores numéricos (sin incluir el límite menor)
  - **maxInclusive**: especifica los límites superiores para los valores numéricos (incluyendo el límite mayor)
  - **minExclusive**: especifica los límites inferiores para los valores numéricos (incluyendo el límite inferior)

```
<xs:element name="minutos">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="59"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="age" type="tipoAge" />
<xs:simpleType name="tipoAge">
  <xs:restriction base="xs:integer">
    <xs:minExclusive value="0"/>
    <xs:maxExclusive value="120"/>
  </xs:restriction>
</xs:simpleType>
```

- **pattern**: define la secuencia exacta de caracteres que son
  - <xs:pattern value="[A-Z][A-Z][A-Z][0-9][0-9][0-9]"/> ⟶ 3 letras mayúsculas y 3 números
  - <xs:pattern value="[a-zA-Z][a-zA-Z][xyz]"/> ⟶ 2 letras mayúsculas o minúsculas y una entre xyz
  - <xs:pattern value="([a-z])*"/> ⟶ 0 ó más letras minúsculas
  - <xs:pattern value="([a-z])+"/> ⟶ 1 ó más letras minúsculas
  - <xs:pattern value="[a-zA-Z0-9]{10}"/> ⟶ 10 caracteres (número, letra mayúscula o minúscula)

```
<xs:element name="initials">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

- Las **extensiones** sirven para derivar tipos a partir de otros tipos de datos. Por ejemplo: creamos un nuevo tipo complejo (tipoUniforme) a partir del anterior, el tipo base tallaCamiseta:

```
<miUniforme colorPantalón="rojo">S</miUniforme>

<xs:schema>
  <xs:element name="miUniforme" type="miUniforme"/>

  <xs:simpleType name="tallaCamiseta">
    <xs:restriction base="xs:string">
      <xs:enumeration value="S"/>
      <xs:enumeration value="M"/>
      <xs:enumeration value="L"/>
      <xs:enumeration value="XL"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="miUniforme">
    <xs:simpleContent>
```

```
            <xs:extension base="tallaCamiseta">
                <xs:attribute name="colorPantalón">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="rojo" />
                            <xs:enumeration value="azul" />
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:schema>
```

- Uso de **simpleContent**: cuando un elemento **complejo solo contiene texto** (es decir, tiene un atributo y en su interior, texto) ⟶ hay que definir su extension (que será base="xs:string) y su atributo

```
<pedido >
    <producto num="10"> Cuaderno </producto>
</pedido>

<xs:schema>
<xs:element name="pedido" type="tipoPedido" />

<xs:complexType name="tipoPedido">
    <xs:sequence>
        <xs:element name="producto" minOccurs="1" maxOccurs="unbounded" type="tipoProducto" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="tipoProducto">
    <xs:simpleContent> ⟶ hay que decir que su contenido es simple
        <xs:extension base="xs:string"> ⟶ hay que decir que la base de su contenido extiende del tipo string
            <xs:attribute name="num" type="xs:positiveInteger" use="optional" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
</xs:schema>
```

- Elementos complejos con **contenido mixto**: Cuando los elementos complejos no tienen un contenido simple, es decir, están formados por elementos, atributos y texto ⟶ <xs:complexType mixed="true">

```
<letter>
    Dear Mr. <name>John Smith</name>. Your order
    <orderid>1032</orderid>will be shipped on
    <shipdate>2001-07-13</shipdate>.
</letter>
```

```
<xs:element name="letter">
    <xs:complexType mixed="true">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="orderid"
            type="xs:positiveInteger"/>
            <xs:element name="shipdate" type="xs:date"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- Indicadores XSD: Nos permiten controlar cómo se utilizan los elementos dentro de los documentos XML
  - Indicadores de **orden**:
    - **All**. Los elementos hijos pueden aparecer en cualquier orden. Cada elemento solo puede aparecer una vez.

```
<xs:element name="person">
 <xs:complexType>
  <xs:all>
   <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
  </xs:all>
 </xs:complexType>
</xs:element>
```

    - **Choice**: Solamente puede aparecer uno de los elementos hijos.

```
<xs:element name="person">
 <xs:complexType>
  <xs:choice>
   <xs:element name="employee" type="employee"/>
   <xs:element name="member" type="member"/>
  </xs:choice>
 </xs:complexType>
</xs:element>
```

    - **Sequence:** Los elementos hijos deben aparecer en el orden especificado.

```
<xs:element name="person">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

  - Indicadores de **ocurrencias:** permiten indicar el número de veces que puede aparecer en elemento.
    - **maxOccurs:** especifica el número **máximo** de veces que puede aparecer un elemento

```
<xs:element name="person">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="fullName" type="xs:string"/>
   <xs:element name="child" type="xs:string" maxOccurs="10"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

→ el elemento child puede aparecer mínimo una vez (el valor predet. para minOccurs es 1) y un máximo de 10 en el elemento person.

    - **minOccurs**: especifica el número **mínimo** de veces que puede aparecer un elemento

```
<xs:element name="person" maxOccurs="unbounded"> --> no hay límite de veces para person
 <xs:complexType>
  <xs:sequence>
   <xs:element name="full_name" type="xs:string"/>
   <xs:element name="child_name" type="xs:string" maxOccurs="10" minOccurs="0"/>
  </xs:sequence>
```

```
  </xs:complexType>
 </xs:element>
```

- **unbounded**: especifica que no hay límite de veces.

- **Elemento <any>** nos permite extender el documento XML con elementos no especificados por el esquema.

```
<xs:element name="person">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
   <xs:any minOccurs="0"/> --> después del elemento lastname podemos añadir cualquier otro
  </xs:sequence>
 </xs:complexType>
</xs:element>

<person>
 <firstname>Hege</firstname>
 <lastname>Refsnes</lastname>
 <children> --> ejemplo de un elemento que podría ser validado con ese elemento "any"
  <childname>Cecilie</childname>
 </children>
</person>
```

- **Atributo anyAttribute** es igual que el elemento any pero para atributo

```
<xs:element name="person">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
  <xs:anyAttribute/>
 </xs:complexType>
</xs:element>

<person eyecolor="blue">
 <firstname>Stale</firstname>
 <lastname>Refsnes</lastname>
</person>
```