

Tema 04 - DTD

Parte 2

9. DTD's

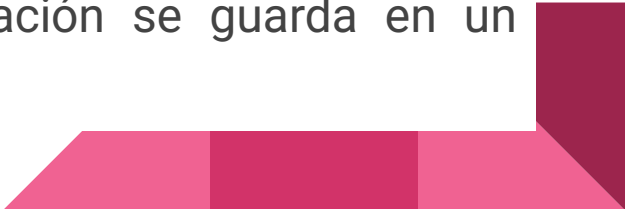
En una empresa tenemos dos programadores que programan así:

<u>Programador 1</u>	<u>Programador 2</u>
<pre><clientes> <cliente> <nombre>AcerSA</nombre> <cif>455321</cif> </cliente> <cliente> <nombre>ACME</nombre> <cif>455321</cif> </cliente> </clientes></pre>	<pre><clientes> <cliente> <cif>455321</cif> <nombre>AcerSA</nombre> </cliente> <cliente> <cif>455321</cif> <nombre>ACME</nombre> </cliente> </clientes></pre>

9. DTD's

Está claro, que ninguno de los dos puede leer los archivos del otro. Por eso, resulta crítico ponerse de acuerdo en lo que se puede hacer, lo que puede aparecer y en qué orden debe hacerlo. Esto se hará mediante las DTD.

DTD significa Declaración de Tipo de Documento, y es un conjunto de reglas sintácticas para definir etiquetas. Nos indica qué etiquetas se pueden usar en un documento, en qué orden deben aparecer, cuáles pueden aparecer dentro de otras, cuáles tienen atributos, etc. Toda esta información se guarda en un archivo de texto con extensión .dtd.



9. DTD's

Un DTD:

- No es un documento XML y por lo tanto no necesita prólogo, salvo en ciertos casos especiales.
- Crear un DTD es como crear nuestro propio lenguaje de marcado para una aplicación específica. Pueden ser parte del documento XML, pero se suele colocar aparte para poder reutilizarlo.
- Puesto que XML es un sistema para definir lenguajes, quien necesite usar XML para intercambio de datos debe definir su propio DTD.
- Un problema que presentan los DTDs es que no siguen la sintaxis XML, sino una propia.

- Supongamos que en nuestros ficheros deseamos indicar que el elemento raíz es <listaclientes>.
- Dentro de <listaclientes> deseamos permitir uno o más elementos <cliente>.
- Dentro de <cliente> todos deberán tener <cif> y <nombre> y en ese orden.
- Dentro de <cliente> puede aparecer o no un elemento <diasentrega> para indicar que ese cliente exige un máximo de plazos.
- Como no todo el mundo usa plazos el <diasentrega> es optativo.
- Se puede indicar si un elemento aparece o no de forma opcional (usando ?)

¿Cuáles son válidos y cuáles no?

```
<listaclientes>
  <cliente>
    <cif>5676443</cif>
    <nombre>Mercasa</nombre>
  </cliente>
</listaclientes>
```

```
<listaclientes>
  <cliente>
    <cif>5676443</cif>
    <diasentrega>30</diasentrega>
  </cliente>
</listaclientes>
```

```
<listaclientes>
  <cliente>
    <cif>5676443</cif>
    <nombre>Mercasa</nombre>
    <diasentrega>30</diasentrega>
  </cliente>
  <cliente>
    <cif>5121554</cif>
    <nombre>Acer SL</nombre>
  </cliente>
</listaclientes>
```

```
<listaclientes>
  <cliente>
    <cif>5676443</cif>
    <nombre>Mercasa</nombre>
    <diasentrega>30</diasentrega>
  </cliente>
</listaclientes>
```

```
<listaclientes>
</listaclientes>
```

```
<listaclientes>
  <cliente>
    <nombre>Mercasa</nombre>
    <cif>5676443</cif>
  </cliente>
  <cliente>
    <cif>5121554</cif>
    <nombre>Acer SL</nombre>
  </cliente>
</listaclientes>
```

10. Definición de elementos en un DTD

Crear el DTD

Para ello creamos un fichero de texto con extensión .dtd.

Vamos describiendo cada elemento uno a uno desde el nodo raíz hasta el último elemento.



10. Definición de elementos en un DTD

Declaración DOCTYPE

Esta declaración es necesaria en el documento XML. Sirve para especificar a que DTD está unido el documento. Siempre debe ir colocado después del prólogo y antes del nodo raíz.

Hay tres formas de especificar un DTD:

- **SYSTEM:** Se utiliza para especificar un DTD "local" o "privado" para especificar un fichero local:

```
<!DOCTYPE nodo_raiz SYSTEM "fichero.dtd">
```

- **PUBLIC:** Se utiliza en el caso de una DTD pública:

```
<!DOCTYPE nodo_raiz PUBLIC "-//OASIS//DTD Libro XML//EN" "../dtds/cap.dtd">
```

- También es posible incrustar la DTD internamente en el documento XML:

10. Definición de elementos en un DTD

DTD incrustado en el documento XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE capitulo
  [<!ELEMENT capitulo (titulo, apartado+)>
   <!ELEMENT titulo (#PCDATA)>
   <!ELEMENT apartado (#PCDATA)>
  ]>
<capitulo>
  <titulo>Introducción</titulo>
  <apartado>Primeros pasos con DTDs.</apartado>
</capitulo>
```

10. Definición de elementos en un DTD

Declaración de elementos

- Los elementos permitidos se especifican con ELEMENT, seguido del nombre y el tipo de elemento:

```
<!ELEMENT nombre_elemento tipo_elemento>
```

- **EMPTY**: significa que el elemento es vacío, es decir, que no puede tener contenido.
- **(#PCDATA)**: significa que el elemento puede contener texto. #PCDATA debe escribirse entre paréntesis.
- **ANY**: significa que el elemento puede contener cualquier cosa (texto y otros elementos). ANY debe escribirse sin paréntesis.

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
>
```

¿Cuáles de las siguientes etiquetas son correctas y cuáles no?

`<ejemplo></ejemplo>`

`<ejemplo />`

`<ejemplo>Esto es un ejemplo</ejemplo>`

`<ejemplo><a></ejemplo>`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?



`<ejemplo></ejemplo>`



`<ejemplo />`



`<ejemplo>Esto es un ejemplo</ejemplo>`

`<!-- ERROR: contiene texto -->`



`<ejemplo><a></ejemplo>`

`<!-- ERROR: contiene un elemento <a> -->`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (#PCDATA)>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

```
<ejemplo />
```

```
<ejemplo>Esto es un ejemplo</ejemplo>
```

```
<ejemplo><a></a></ejemplo>
```

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (#PCDATA)>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?



```
<ejemplo />
```



```
<ejemplo>Esto es un ejemplo</ejemplo>
```



```
<ejemplo><a></a></ejemplo>
```

```
<!-- ERROR: contiene un elemento <a> -->
```

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo ANY>  
  <!ELEMENT a ANY>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?

`<ejemplo />`

`<ejemplo>Esto es un ejemplo</ejemplo>`

`<ejemplo>Esto es <a>un ejemplo</ejemplo>`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo ANY>  
  <!ELEMENT a ANY>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?



<ejemplo />



<ejemplo>Esto es un ejemplo</ejemplo>



<ejemplo>Esto es <a>un ejemplo</ejemplo>

10. Definición de elementos en un DTD

Declaración de elementos

- Para indicar que un elemento puede o debe contener otros elementos se deben indicar los elementos, utilizando los conectores y modificadores siguientes:

- **, (coma)**: significa que el elemento contiene los elementos en el orden indicado.

```
<!ELEMENT libro (titulo, autor, capitulo)>
```

libro debe contener un titulo, un autor y un capitulo exactamente y por ese orden.

- **| (o lógico)**: significa que el elemento contiene uno de los dos elementos.

```
<!ELEMENT libro (titulo | autor | capitulo)>
```

libro debe contener un titulo, un autor o un capitulo

- **? (opcional)**: significa que el elemento puede aparecer o no, pero sólo una vez.
- *****: significa que el elemento puede aparecer o no una o más veces.
- **+ (obligatorio)**: significa que el elemento tiene que aparecer una o más veces (no puede no aparecer).
- **()**: permite agrupar expresiones.

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (a, b)>  
  <!ELEMENT a EMPTY>  
  <!ELEMENT b EMPTY>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

`<ejemplo><a /></ejemplo>`

`<ejemplo><a /><c /></ejemplo>`

`<ejemplo><a /></ejemplo>`

`<ejemplo><a /></ejemplo>`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (a, b)>  
  <!ELEMENT a EMPTY>  
  <!ELEMENT b EMPTY>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?



<ejemplo><a /></ejemplo>



<ejemplo><a /><c /></ejemplo>

<!-- ERROR: contiene un elemento <c /> -->



<ejemplo><a /></ejemplo>

<!-- ERROR: falta el elemento -->



<ejemplo><a /></ejemplo>

<!-- ERROR: el orden no es correcto -->

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (a | b)>  
  <!ELEMENT a EMPTY>  
  <!ELEMENT b EMPTY>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

<ejemplo><a /></ejemplo>

<ejemplo></ejemplo>

<ejemplo><a /></ejemplo>

<ejemplo></ejemplo>

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (a | b)>  
  <!ELEMENT a EMPTY>  
  <!ELEMENT b EMPTY>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?



```
<ejemplo><a /></ejemplo>
```



```
<ejemplo><b /></ejemplo>
```



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

```
<!-- ERROR: están los dos elementos -->
```



```
<ejemplo></ejemplo>
```

```
<!-- ERROR: no hay ningún elemento -->
```

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (a, b?)>  
  <!ELEMENT a EMPTY>  
  <!ELEMENT b EMPTY>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

<ejemplo><a /></ejemplo>

<ejemplo><a /></ejemplo>

<ejemplo></ejemplo>

<ejemplo></ejemplo>

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (a, b?)>  
  <!ELEMENT a EMPTY>  
  <!ELEMENT b EMPTY>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?



<ejemplo><a /></ejemplo>



<ejemplo><a /></ejemplo>



<ejemplo></ejemplo>

<!-- ERROR: falta el elemento <a /> -->



<ejemplo></ejemplo>

<!-- ERROR: el elemento aparece dos veces -->

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (a*, b)>  
  <!ELEMENT a EMPTY>  
  <!ELEMENT b EMPTY>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

<ejemplo></ejemplo>

<ejemplo><a /></ejemplo>

<ejemplo><a /><a /></ejemplo>

<ejemplo><a /></ejemplo>

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (a*, b)>  
  <!ELEMENT a EMPTY>  
  <!ELEMENT b EMPTY>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?



```
<ejemplo><b /></ejemplo>
```



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



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



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

```
<!-- ERROR: el elemento <a /> aparece después de <b /> -->
```

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (a+, b)>  
  <!ELEMENT a EMPTY>  
  <!ELEMENT b EMPTY>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?

`<ejemplo><a /></ejemplo>`

`<ejemplo><a /><a /></ejemplo>`

`<ejemplo></ejemplo>`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (a+, b)>  
  <!ELEMENT a EMPTY>  
  <!ELEMENT b EMPTY>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?



<ejemplo><a /></ejemplo>



<ejemplo><a /><a /></ejemplo>



<ejemplo></ejemplo>

<!-- ERROR: falta el elemento <a /> -->

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (a, (a|b))>  
  <!ELEMENT a EMPTY>  
  <!ELEMENT b EMPTY>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

`<ejemplo><a /><a /></ejemplo>`

`<ejemplo><a /></ejemplo>`

`<ejemplo><a /></ejemplo>`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (a, (a|b))>  
  <!ELEMENT a EMPTY>  
  <!ELEMENT b EMPTY>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?



`<ejemplo><a /><a /></ejemplo>`



`<ejemplo><a /></ejemplo>`



`<ejemplo><a /></ejemplo>`

`<!-- ERROR: falta el elemento <a /> o -->`

Ejemplo

Unos programadores necesitan un formato de fichero para que sus distintos programas intercambien información sobre ventas. El acuerdo al que han llegado es que su XML debería tener esta estructura:

- El elemento raíz será <listaventas>
- Toda <listaventas> tiene una o más ventas.
- Toda <venta> tiene los siguientes datos:
- Importe.
- Comprador.
- Vendedor.
- Fecha (optativa).
- Un código de factura.

```
<listaventas>
  <venta>
    <importe>1500</importe>
    <comprador>Wile E.Coyote</comprador>
    <vendedor>ACME</vendedor>
    <codigofactura>E17</codigofactura>
  </venta>
  <venta>
    <importe>750</importe>
    <comprador>Elmer Fudd</comprador>
    <vendedor>ACME</vendedor>
    <fecha>27-2-2015</fecha>
    <codigofactura>E18</codigofactura>
  </venta>
</listaventas>
```

Possible solution:

```
<!DOCTYPE listventas[
  <!ELEMENT listventas (venta+)>
  <!ELEMENT venta (importe, comprador,
vendedor, fecha?, codigofactura)>
  <!ELEMENT importe (#PCDATA)>
  <!ELEMENT comprador (#PCDATA)>
  <!ELEMENT vendedor (#PCDATA)>
  <!ELEMENT fecha (#PCDATA)>
  <!ELEMENT codigofactura (#PCDATA)>

]>
```

10. Definición de elementos en un DTD

Declaración de atributos

Los atributos permitidos para un elemento se especifican con ATTLIST y sigue la siguiente sintaxis:

```
<!ATTLIST nombreElemento nombreAtributo tipoAtributo valorInicialAtributo >
```

Donde:

- **nombreElemento:** es el nombre del elemento para el que se define un atributo.
- **nombreAtributo:** es el nombre del atributo.
- **tipoAtributo:** es el tipo de datos.
- **valorInicialAtributo:** es el valor predeterminado del atributo (aunque también puede indicar otras cosas).

10. Definición de elementos en un DTD

Declaración de atributos

¡Importante! Para definir varios atributos de un mismo elemento, se puede utilizar una o varias declaraciones de atributos.

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



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

10. Definición de elementos en un DTD

Declaración de atributos

Los tipos de definiciones de atributos que podemos encontrarnos son:

- **CDATA:** el atributo contiene caracteres (sin restricciones).
- **NMTOKEN:** el atributo sólo contiene letras, dígitos, y los caracteres ".", "-", "_" y ":".
- **NMTOKENS:** el atributo sólo contiene letras, dígitos, y los caracteres ".", "-", "_", ":" (como el tipo NMTOKEN) y también espacios en blanco.
- **valores concretos:** el atributo sólo puede contener uno de los términos de una lista. La lista se escribe entre paréntesis, con los términos separados por una barra vertical "|".
- **ID:** el valor del atributo (no el nombre) debe ser único y no se puede repetir en otros elementos o atributos.

10. Definición de elementos en un DTD

Declaración de atributos

Los tipos de definiciones de atributos que podemos encontrarnos son:

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

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color CDATA #REQUIRED>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

<ejemplo color="" />

<ejemplo color="amarillo" />

<ejemplo color="azul marino #000080" />

<ejemplo />

<ejemplo sabor="dulce" />

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color CDATA #REQUIRED>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?



`<ejemplo color="" />`



`<ejemplo color="amarillo" />`



`<ejemplo color="azul marino #000080" />`



`<ejemplo />`

`<!-- ERROR: falta el atributo "color", obligatorio debido al #REQUIRED -->`



`<ejemplo sabor="dulce" />`

`<!-- ERROR: el atributo "sabor" no está definido -->`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color NMTOKEN #REQUIRED>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

`<ejemplo color="" />`

`<ejemplo color="azul-marino" />`

`<ejemplo color="1" />`

`<ejemplo color="azul marino" />`

`<ejemplo color="#F0F0F0" />`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color NMTOKEN #REQUIRED>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?



`<ejemplo color="" />`



`<ejemplo color="azul-marino" />`



`<ejemplo color="1" />`



`<ejemplo color="azul marino" />`

`<!-- ERROR: hay un espacio en blanco -->`



`<ejemplo color="#F0F0F0" />`

`<!-- ERROR: contiene el carácter # -->`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color NMTOKENS #REQUIRED>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?

`<ejemplo color="" />`

`<ejemplo color="1" />`

`<ejemplo color="azul marino" />`

`<ejemplo color="2*2" />`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color NMTOKENS #REQUIRED>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?



```
<ejemplo color="" />
```



```
<ejemplo color="1" />
```



```
<ejemplo color="azul marino" />
```



```
<ejemplo color="2*2" />
```

```
<!-- ERROR: hay un asterisco -->
```

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color (azul|blanco|rojo) #REQUIRED>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

`<ejemplo color="" />`

`<ejemplo color="azul" />`

`<ejemplo color="verde" />`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color (azul|blanco|rojo) #REQUIRED>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?



```
<ejemplo color="" />
```



```
<ejemplo color="azul" />
```



```
<ejemplo color="verde" />
```

```
<!-- ERROR: "verde" no está en la lista de valores -->
```

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (libro*)>  
  <!ELEMENT libro (#PCDATA) >  
  <!ATTLIST libro codigo ID #REQUIRED>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>  
</ejemplo>
```

```
<ejemplo>  
  <libro codigo="1">Poema de Gilgamesh</libro>  
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>  
</ejemplo>
```

```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <libro codigo="L1">Los preceptos de Ptah-Hotep</libro>  
</ejemplo>
```

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo (libro*)>  
  <!ELEMENT libro (#PCDATA) >  
  <!ATTLIST libro codigo ID #REQUIRED>  
>]
```

¿Cuál de los siguientes elementos son correctos y cuáles no?



```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>  
</ejemplo>
```



```
<ejemplo>  
  <libro codigo="1">Poema de Gilgamesh</libro>  
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>  
</ejemplo>
```

<!-- ERROR: el valor de un atributo de tipo ID no puede empezar con un número -->



```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <libro codigo="L1">Los preceptos de Ptah-Hotep</libro>  
</ejemplo>
```

<!-- ERROR: no se puede repetir un atributo de tipo ID -->

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo ((libro|prestamo)*)>  
  <!ELEMENT libro (#PCDATA) >  
  <!ATTLIST libro codigo ID #REQUIRED>  
  <!ELEMENT prestamo (#PCDATA) >  
  <!ATTLIST prestamo libro IDREF #REQUIRED>  
>]
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <prestamo libro="L1">Numa Nigerio</prestamo>  
</ejemplo>
```

```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <prestamo libro="L2">Numa Nigerio</prestamo>  
</ejemplo>
```

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo ((libro|prestamo)*)>  
  <!ELEMENT libro (#PCDATA) >  
  <!ATTLIST libro codigo ID #REQUIRED>  
  <!ELEMENT prestamo (#PCDATA) >  
  <!ATTLIST prestamo libro IDREF #REQUIRED>  
>]
```

¿Cuál de los siguientes elementos son correctos y cuáles no?



```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <prestamo libro="L1">Numa Nigerio</prestamo>  
</ejemplo>
```



```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <prestamo libro="L2">Numa Nigerio</prestamo>  
</ejemplo>
```

<!-- ERROR: el valor "L2" no es ID de ningún elemento -->

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo ((libro|prestamo)*)>  
  <!ELEMENT libro (#PCDATA) >  
  <!ATTLIST libro codigo ID #REQUIRED>  
  <!ELEMENT prestamo (#PCDATA) >  
  <!ATTLIST prestamo libro IDREFS #REQUIRED>  
>]
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>  
  <prestamo libro="L1 L2">Numa Nigerio</prestamo>  
</ejemplo>
```


```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>  
  <prestamo libro="L1">Numa Nigerio</prestamo>  
</ejemplo>
```

```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>  
  <prestamo libro="L3">Numa Nigerio</prestamo>  
</ejemplo>
```



Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo ((libro|prestamo)*)>  
  <!ELEMENT libro (#PCDATA) >  
  <!ATTLIST libro codigo ID #REQUIRED>  
  <!ELEMENT prestamo (#PCDATA) >  
  <!ATTLIST prestamo libro IDREFS #REQUIRED>  
>]
```


¿Cuál de los siguientes elementos son correctos y cuáles no?



```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>  
  <prestamo libro="L1 L2">Numa Nigerio</prestamo>  
</ejemplo>
```



```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>  
  <prestamo libro="L1">Numa Nigerio</prestamo>  
</ejemplo>
```



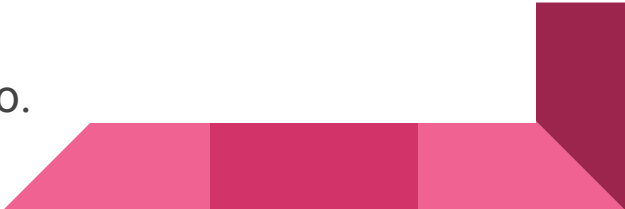
```
<ejemplo>  
  <libro codigo="L1">Poema de Gilgamesh</libro>  
  <libro codigo="L2">Los preceptos de Ptah-Hotep</libro>  
  <prestamo libro="L3">Numa Nigerio</prestamo>  
</ejemplo>
```

<!-- ERROR: el valor "L3" no es ID de ningún elemento -->

10. Definición de elementos en un DTD

Declaración de atributos

Por otra parte, los atributos pueden tomar valores iniciales, que son los siguientes:

- **#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 concreto:** el atributo tiene un valor predeterminado.
- 

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color CDATA #REQUIRED>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?

`<ejemplo color="" />`

`<ejemplo color="amarillo" />`

`<ejemplo color="azul marino #000080" />`

`<ejemplo />`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color CDATA #REQUIRED>  

```

¿Cuál de los siguientes elementos son correctos y cuáles no?



```
<ejemplo color="" />
```



```
<ejemplo color="amarillo" />
```



```
<ejemplo color="azul marino #000080" />
```



```
<ejemplo />
```

```
<!-- ERROR: falta el atributo "color" -->
```

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color CDATA #IMPLIED>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

<ejemplo />

<ejemplo color="" />

<ejemplo color="amarillo" />

<ejemplo color="azul marino #000080" />

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color CDATA #IMPLIED>  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?



`<ejemplo />`



`<ejemplo color="" />`



`<ejemplo color="amarillo" />`



`<ejemplo color="azul marino #000080" />`

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color CDATA #FIXED "verde">  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

<ejemplo />

<ejemplo color="verde" />

<ejemplo color="" />

<ejemplo color="amarillo" />

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color CDATA #FIXED "verde">  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?



<ejemplo />



<ejemplo color="verde" />



<ejemplo color="" />

<!-- ERROR: el atributo "color" no tiene el valor "verde" -->



<ejemplo color="amarillo" />

<!-- ERROR: el atributo "color" no tiene el valor "verde" -->

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color CDATA "verde">  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?

<ejemplo />

<ejemplo color="" />

<ejemplo color="amarillo" />

<ejemplo color="verde" />

Dado

```
<!DOCTYPE ejemplo [  
  <!ELEMENT ejemplo EMPTY>  
  <!ATTLIST ejemplo color CDATA "verde">  
>
```

¿Cuál de los siguientes elementos son correctos y cuáles no?



<ejemplo />



<ejemplo color="" />



<ejemplo color="amarillo" />



<ejemplo color="verde" />

Tema 04 - XSD

Parte 3

11. XML Schema

XML Schema es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Se consigue así una percepción del tipo de documento con un nivel alto de abstracción. Fue desarrollado por el World Wide Web Consortium (W3C) y alcanzó el nivel de recomendación en mayo de 2001.

XML Schema está pensado para proporcionar una mayor potencia expresiva que las DTD, menos capaces al describir los documentos a nivel formal.



11. XML Schema

Ventajas que aporta XML Schema frente a DTD

- **Están escritos en XML:** Por lo tanto, las mismas bibliotecas que permiten procesar ficheros XML de datos permitirían procesar ficheros XML de reglas.
- **Son mucho más potentes:** ofrecen soporte a tipos de datos con comprobación de si el contenido de una etiqueta es de tipo integer, date o de otros tipos. También se permite añadir restricciones como indicar valores mínimo y máximo para un número o determinar el patrón que debe seguir una cadena válida
- **Ofrecen la posibilidad de usar espacios de nombres:** Los espacios de nombres son similares a los paquetes Java: permiten a personas distintas el definir etiquetas con el mismo nombre pudiendo luego distinguir etiquetas iguales en función del espacio de nombres que importemos.

12. Estructura de un XML Schema

Utilización del esquema

Para utilizar el esquema desde un documento XML, tenemos que tener en cuenta si está en nuestro sistema de ficheros local o es un esquema público.

- En caso de que el esquema esté en un sitio público la etiqueta del nodo raíz cambia a:

```
<nodo_raiz xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.miempresa.com/mi_esquema.xsd">
```

- En caso de que el esquema esté en local la etiqueta del nodo raíz cambia a:

```
<nodo_raiz xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="mi_esquema.xsd">
```

12. Estructura de un XML Schema

Nodo raíz → schema

Los esquemas mantienen la estructura y la sintaxis de los documentos XML por lo tanto **deben tener un nodo raíz y dentro de él contener todo el esquema**. Por supuesto debe tener el prólogo (es decir, la definición de la versión XML).

```
<?xml version="1.0" encoding="UTF-8"?>  
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  ...  
</xsd:schema>
```



12. Estructura de un XML Schema

Nodo raíz → schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  ...
</xs:schema>
```

Utiliza un atributo `xmlns` que significa XML namespace para crear un espacio de nombres de XML vinculado a un prefijo que en este caso es `xsd`, pero que puede ser cualquier cosa ya que sólo hace la función de variable (también se usa `xs`).

Hace referencia a una URL que contiene la definición de todos los elementos y atributos que se pueden utilizar en un esquema. **Eso no quiere decir que para programar en XML se necesite estar conectado a Internet.**

12. Estructura de un XML Schema

Nodo raíz → schema

Por ejemplo, supongamos que deseamos tener ficheros XML con un solo elemento llamado <cantidad> que debe tener dentro un número.

XML	XML Schema
<code><cantidad>20</cantidad></code>	<pre><xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"> <xs:element name="cantidad" type="xs:integer"/> </xs:schema></pre>

12. Estructura de un XML Schema

¿Qué contiene este fichero?

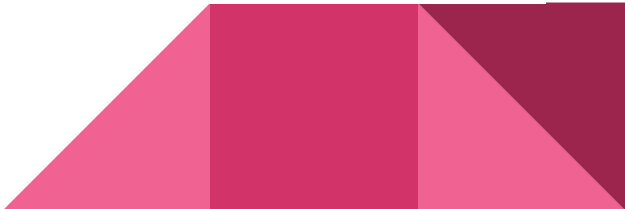
1. Se indica que este fichero va a usar unas etiquetas ya definidas en un espacio de nombres (o **XML Namespace**, de ahí **xmlns**). Esa definición se hace en el espacio de nombres que aparece en la URL. Nuestro validador no descargará nada, esa URL es oficial y todos los validadores la conocen. **Las etiquetas de ese espacio de nombres van a usar un prefijo que en este caso será xs**. Nótese que el prefijo puede ser como queramos (podría ser «abcd» o «zztop»), pero la costumbre es usar **xsd** ó **xs**.
2. Se indica que habrá un solo elemento y que el tipo de ese elemento es **<xsd:integer>**. Es decir, un entero básico.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <xs:element name="cantidad" type="xsd:integer"/>  
</xs:schema>
```

12. Estructura de un XML Schema

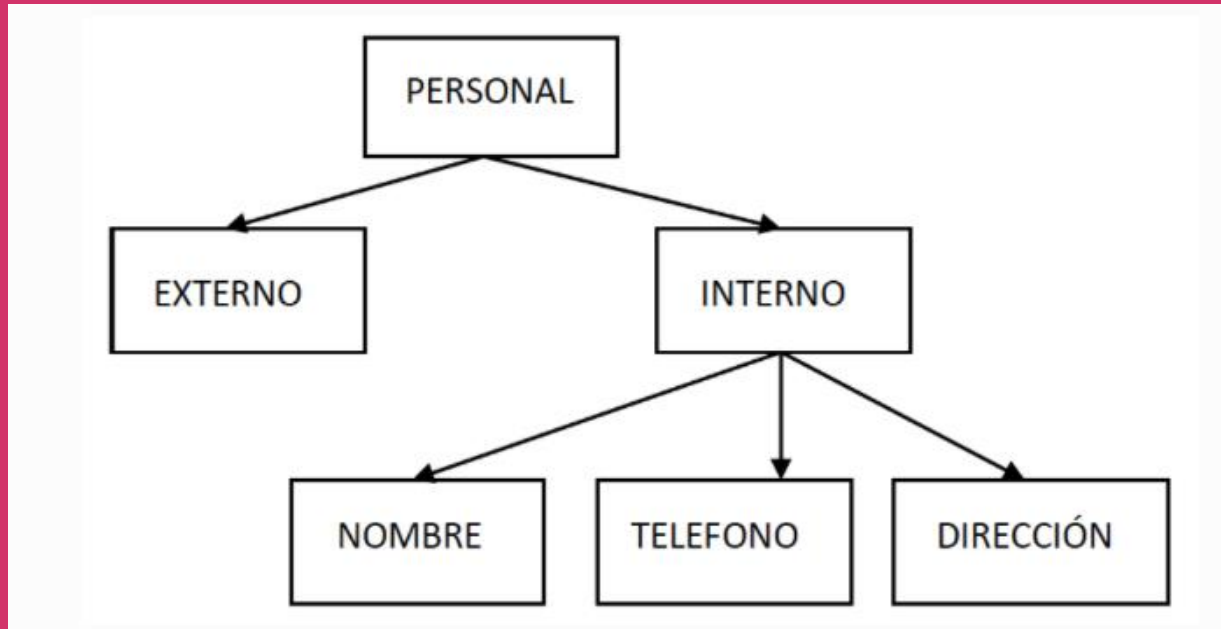
Declaración de elementos

Existen tres tipos de elementos a definir en un esquema:

- **Elementos estándar:** Sin hijos y sin restricciones ni atributos.
 - **Elementos simples:** Elementos sin hijos pero con restricciones y sin atributos.
 - **Elementos complejos:** Elementos con hijos y/o atributos.
- 

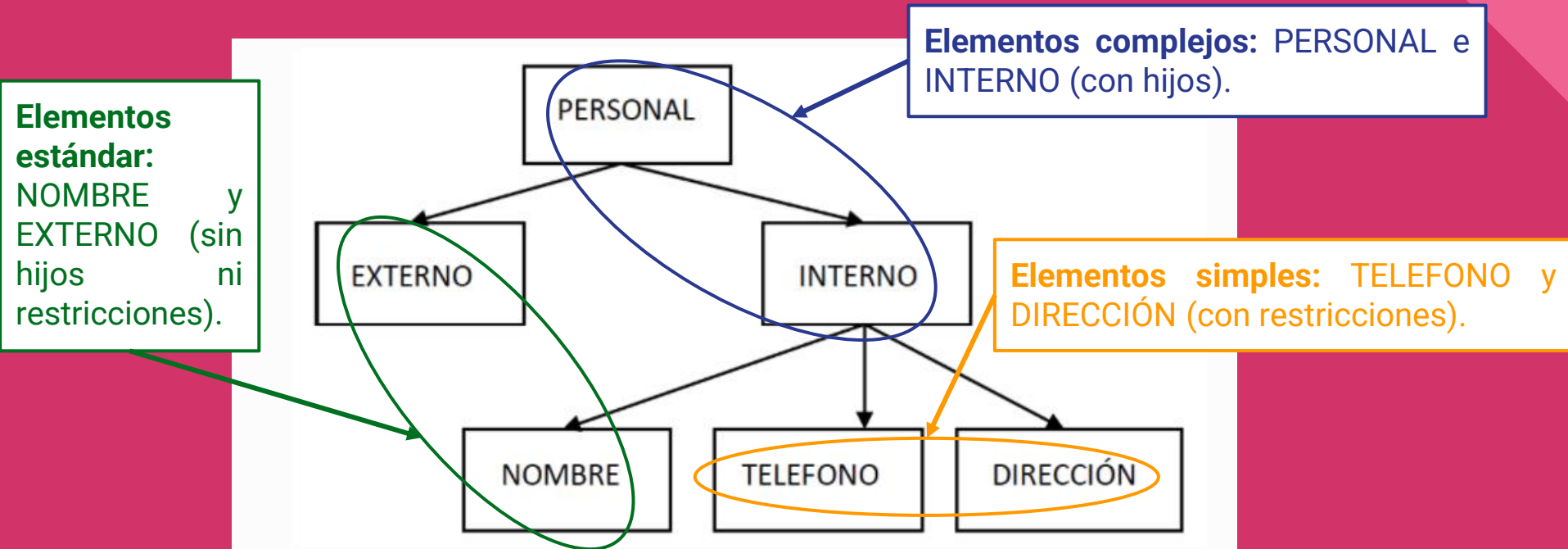
Ejemplo

Identifica cada tipo de elemento en la siguiente jerarquía sabiendo que teléfono y dirección cuentan con restricciones.



Ejemplo

Identifica cada tipo de elemento en la siguiente jerarquía



12. Estructura de un XML Schema

Estructura de un elemento

```
<xs:element name="nombre_del_elemento" type="tipo" />
```

En esta estructura se define un nombre de elemento y un tipo, y el tipo puede ser un estándar de XML o un tipo definido por nosotros que describiría aquellos elementos que tengan hijos (complejos o aquellos elementos que no tienen hijos pero tienen restricciones).

Cuando es un tipo complejo, seguido de esta definición, vendría debajo la definición del tipo. Esta estructura es la más aconsejable pero no la única.

12. Estructura de un XML Schema

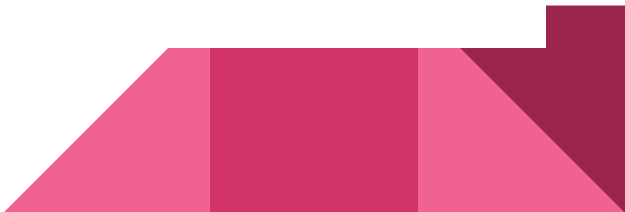
Estructura de un elemento

```
<xs:element name="nombre_del_elemento">
```

...

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

Esta estructura sería equivalente a la anterior sólo que en este caso no se pone el tipo del elemento como atributo sino que esa definición se haría entre las etiquetas de inicio y cierre de element.



12. Estructura de un XML Schema

Estructura de un elemento

Un elemento puede contener atributos cuyos valores siempre van entre comillas:

- **name:** Nombre del elemento.
- **type:** Tipo simple predefinido, ya sean los estándares o unos propios.
- **maxOccurs:** Número máximo de veces que puede aparecer [0..unbounded].
- **minOccurs:** Número mínimo de veces que puede aparecer.
- **ref:** Para importar de otros esquemas o hacer referencia a un elemento ya declarado anteriormente en este mismo esquema.

Ejemplo

Dado el siguiente XML y DTD, define los elementos AUTOR, TITULO, EDITORIAL Y ANIO para que pertenezcan a un nuevo Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE BIBLIOTECA SYSTEM "biblioteca.dtd">
<BIBLIOTECA>
  <LIBRO COD="1">
    <TITULO>XML PARA TODOS</TITULO>
    <AUTOR>SERGIO LUJAN</AUTOR>
    <ANIO>2001</ANIO>
    <EDITORIAL>UA PRENSA</EDITORIAL>
  </LIBRO>
  <LIBRO COD="11">
    <TITULO>Como aprobar una oposicion</TITULO>
    <AUTOR>Marisa Zayas</AUTOR>
    <ANIO>1999</ANIO>
    <EDITORIAL>Prensa Editorial</EDITORIAL>
  </LIBRO>
</BIBLIOTECA>
```

```
<ELEMENT BIBLIOTECA (LIBRO+)>
<ELEMENT LIBRO (TITULO, AUTOR*, ANIO?, EDITORIAL?)>
<ATTLIST LIBRO COD CDATA #REQUIRED>
<ELEMENT AUTOR (#PCDATA)>
<ELEMENT TITULO (#PCDATA)>
<ELEMENT EDITORIAL (#PCDATA)>
<ELEMENT ANIO (#PCDATA)>
```

Ejemplo

```
<xs:element name="TITULO" type="xs:string"
  minOccurs="1" maxOccurs="1" />
<xs:element name="AUTOR" type="xs:string"
  minOccurs="0" maxOccurs="unbounded" />
<xs:element name="ANYO" type="xs:string"
  minOccurs="0" maxOccurs="1" />
<xs:element name="EDITORIAL"
  type="xs:string" minOccurs="0"
  maxOccurs="1" />
```

Ejemplo

Otra forma de verlo

```
<xs:element name="TITULO" type="xs:string"
/>
<xs:element name="AUTOR" type="xs:string"
  minOccurs="0" maxOccurs="unbounded" />
<xs:element name="ANYO" type="xs:string"
  minOccurs="0" />
<xs:element name="EDITORIAL"
  type="xs:string" minOccurs="0" />
```

13. Tipo complejo: complexType

Sirve para definir elementos que tienen elementos hijo y/o atributos.

```
<xs:complexType>
```

```
  <xs:sequence> <!-- sequence/all/choice -->
```

```
    ... subelementos ...
```

```
  </xs:sequence>
```

```
  ... atributos ...
```

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

Estos tres elementos nunca se utilizan juntos, **aparece tan sólo uno de ellos en el elemento complexType**. Sirve para describir en qué orden y cómo deben aparecer los subelementos del complexType.

Es equivalente a, en el DTD, poner comas o barras verticales en la descripción de un elemento con hijos.

13. Tipo complejo: complexType

Ejemplo

```
<xs:element name="contacto">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="destinatario" type="xs:string" />
      <xs:element name="remitente" type="xs:string" />
      <xs:element name="titulo" type="xs:string" />
      <xs:element name="contenido" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="fecha" type="xs:date"/>
  </xs:complexType>
</xs:element>
```



13. Tipo complejo: complexType

Puede contener elementos secundarios:	Puede tener los siguientes atributos:
<ul style="list-style-type: none">- sequence: Implica que deben aparecer todos los elementos y en ese orden (AND).- all: Implica que deben aparecer todos los elementos, sin importar el orden.- choice: Implica que sólo debe aparecer uno de esos elementos (OR).- attribute: Para definir atributos.	<ul style="list-style-type: none">- name: Nombre del tipo complejo.- mixed: Puede tener dos valores true o false.- type: Tipo de datos con el que se identifica.

13. Tipo complejo: complexType

Elemento sequence

Este elemento indica que es obligatorio que aparezcan todos los elementos especificados y en el orden en que se definen. Es el equivalente a un AND (**la** , en un DTD).



Ejemplo

Dado el siguiente XML y sabiendo que título, autor y editorial tienen que aparecer obligatoriamente (solo una vez) y en este orden, genera el correspondiente schema.

```
<libro>
```

```
  <titulo>El señor de los anillos</titulo>
```

```
  <autor>John Ronald Ruelen Tolkien</autor>
```

```
  <editorial>Tirant Lo Blanch</editorial>
```

```
</libro>
```


Ejemplo

```
<xs:element name="libro"/>

  <xs:complexType>

    <xs:sequence>

      <xs:element name="titulo" type="xs:string" />

      <xs:element name="autor" type="xs:string" />

      <xs:element name="editorial" type="xs:string" />

    </xs:sequence>

  </xs:complexType>

</xs:element>

<libro>

  <titulo>El señor de los anillos</titulo>

  <autor>John Ronald Ruelen Tolkien</autor>

  <editorial>Tirant Lo Blanch</editorial>

</libro>
```

En este ejemplo se define el elemento libro, con tres subelementos obligatorios y que deben aparecer exactamente en este orden (1º titulo, 2º autor y 3º editorial) y no aparecen es este orden o uno de ellos no aparece, se produciría un error.

13. Tipo complejo: complexType

Elemento all

Este elemento indica que es obligatorio que aparezcan todos los elementos especificados y pero NO en el orden en que se definen.



Ejemplo

Dado el siguiente XML y sabiendo que título, autor y editorial tienen que aparecer obligatoriamente pero no en este orden, genera el correspondiente schema.

```
<libro>
```

```
  <titulo>El señor de los anillos</titulo>
```

```
  <autor>John Ronald Ruelen Tolkien</autor>
```

```
  <editorial>Tirant Lo Blanch</editorial>
```

```
</libro>
```

Ejemplo

```
<xs:element name="libro"/>
```

```
<xs:complexType>  
  <xs:all>  
    <xs:element name="titulo" type="xs:string" />  
    <xs:element name="autor" type="xs:string" />  
    <xs:element name="editorial" type="xs:string" />  
  </xs:all>  
</xs:complexType>  
</xs:element>
```

```
<libro>  
  
  <titulo>El señor de los anillos</titulo>  
  
  <autor>John Ronald Ruelen Tolkien</autor>  
  
  <editorial>Tirant Lo Blanch</editorial>  
  
</libro>
```

En este ejemplo se define el elemento libro, con tres subelementos obligatorios.

13. Tipo complejo: complexType

Elemento choice

Este elemento indica que de todos los elementos especificados sólo debe aparecer uno de ellos. Es el equivalente al OR (**es la | en un DTD**).



Ejemplo

Dado el siguiente XML y sabiendo que de título, autor y editorial sólo debe aparecer uno de ellos, genera el correspondiente schema.

```
<libro>  
  <autor>J.R.R. Tolkien</autor>  
  <titulo>El señor de los anillos</titulo>  
  <editorial>Montena</editorial>  
</libro>
```

Ejemplo

```
<xs:element name="libro" type="tipo_libro"/>

<xs:complexType name="tipo_libro">

  <xs:choice>

    <xs:element name="titulo" type="xs:string" />

    <xs:element name="autor" type="xs:string" />

    <xs:element name="editorial" type="xs:string" />

  </xs:choice>

</xs:complexType>

</xs:element>

<libro>

  <titulo>El señor de los anillos</titulo>

</libro>
```

En este ejemplo se define el elemento libro, con tres posibles subelementos. Puede tener o un título o un autor o una editorial.

13. Tipo complejo: complexType

Elemento attribute

Para definir los atributos de un elemento o tipo de elemento utilizamos la siguiente estructura:

```
<xs:attribute name="nombre_atributo" type="tipo_atributo" use="modificador" />
```

La localización del atributo no puede ir por sí solo, ya que con esta estructura no sabríamos a qué elemento se refiere. Para ello se pone siempre dentro de una estructura complexType.

13. Tipo complejo: complexType

Elemento attribute

```
<xs:attribute name="nombre_atributo" type="tipo_atributo" use="modificador" />
```

name: Es el nombre del atributo.

type: Es el tipo del atributo.

use: Para definir si es un atributo obligatorio u opcional. Para definir un atributo como obligatorio le asignaremos el valor required. Por defecto es opcional.

13. Tipo complejo: complexType

Elemento attribute

Hay algunas reglas que es necesario cumplir cuando se definen atributos:

- Tiene que aparecer después de la declaración de la declaración de los subelementos.
- No puede contener hijos.
- Su declaración no impone un orden de uso.
- Su tipo por defecto es anySimpleType (cualquier cadena de caracteres XML válidos).
- Un atributo sólo puede aparecer una vez en un elemento dado.
- A menos que se especifique el atributo será opcional.
- Existen tres atributos para restringir los valores del atributo: use, default y fixed.

13. Tipo complejo: complexType

Elemento attribute

- **use:** Puede tomar tres valores:
 - ***required***: atributo obligatorio.
 - ***optional***: atributo opcional, puede o no aparecer. Este es el valor por defecto.
 - ***prohibited***: el atributo no puede aparecer en el elemento.
- **default:** define el valor por defecto del atributo. Si el atributo no está en el elemento entonces XML emplea este valor.
- **fixed:** define un valor fijo para el atributo. El atributo puede aparecer o no en el elemento, pero si aparece sólo puede tomar ese valor.

Ejemplo

Dado el siguiente XML y DTD, define su Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE BIBLIOTECA SYSTEM "biblioteca.dtd">
<BIBLIOTECA>
  <LIBRO COD="1">
    <TITULO>XML PARA TODOS</TITULO>
    <AUTOR>SERGIO LUJAN</AUTOR>
    <ANIO>2001</ANIO>
    <EDITORIAL>UA PRENSA</EDITORIAL>
  </LIBRO>
  <LIBRO COD="11">
    <TITULO>Como aprobar una oposicion</TITULO>
    <AUTOR>Marisa Zayas</AUTOR>
    <ANIO>1999</ANIO>
    <EDITORIAL>Prensa Editorial</EDITORIAL>
  </LIBRO>
</BIBLIOTECA>
```

```
<!ELEMENT BIBLIOTECA (LIBRO+)>
<!ELEMENT LIBRO (TITULO, AUTOR*, ANIO?, EDITORIAL?)>
<!-- ATTLIST LIBRO COD CDATA #REQUIRED -->
<!ELEMENT AUTOR (#PCDATA)>
<!ELEMENT TITULO (#PCDATA)>
<!ELEMENT EDITORIAL (#PCDATA)>
<!ELEMENT ANIO (#PCDATA)>
```

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

```
<BIBLIOTECA xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="biblioteca.xsd">
```

```
<LIBRO COD="1">
```

```
<TITULO>XML PARA TODOS</TITULO>
```

```
<AUTOR>SERGIO LUJAN</AUTOR>
```

```
<ANIO>2001</ANIO>
```

```
<EDITORIAL>UA PRENSA</EDITORIAL>
```

```
</LIBRO>
```

```
<LIBRO COD="11">
```

```
<TITULO>Como aprobar una oposicion</TITULO>
```

```
<AUTOR>Marisa Zayas</AUTOR>
```

```
<ANIO>1999</ANIO>
```

```
<EDITORIAL>Prensa Editorial</EDITORIAL>
```

```
</LIBRO>
```

```
</BIBLIOTECA>
```

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
<xs:element name="BIBLIOTECA">
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```
<xs:element name="LIBRO" minOccurs="1" maxOccurs="unbounded">
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```
<xs:element name="TITULO" type="xs:string"/>
```

```
<xs:element name="AUTOR" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
```

```
<xs:element name="ANIO" type="xs:string" minOccurs="0"/>
```

```
<xs:element name="EDITORIAL" type="xs:string" minOccurs="0"/>
```

```
</xs:sequence>
```

```
<xs:attribute name="COD" type="xs:string"/>
```

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

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

```
</xs:sequence>
```

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

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


```
</xs:schema>
```

14. Tipo simple: simpleType

En XML Schema se pueden restringir los valores que pueden tomar los elementos y los atributos. A estas restricciones se las denomina facetas.

`<diaSemana>7</diaSemana>`

*No tiene sentido
que sea mayor que
7*

A red arrow points from the text box to the value '7' in the XML code.

14. Tipo simple: simpleType

Un tipo simple sirve para definir una serie de restricciones a un elemento o a un atributo. Es muy útil para definir rangos, tipos enumerados, etc.

```
<xs:simpleType>
```

```
  <xs:restriction base=""> (o extensión)
```

```
    ... restricciones ...
```

```
  </xs:restriction>
```

```
</xs:simpleType>
```



14. Tipo simple: simpleType

XML Schema permite crear tipos derivados a partir de tipos existentes. Existen dos tipos de mecanismos para la creación o derivación de los tipos de datos:

- Por restricción → `<xs:restriction base=""> </xs:restriction>`
- Por extensión → `<xs:extension base=""> </xs:extension>`

En ambos casos el atributo base es el tipo del que se está derivando.



14. Tipo simple: simpleType - restricción

Elemento restriction

Se utiliza para poner rangos, patrones enumerar posibles valores, etc.

```
<xs:restriction base="xs:string">
```

Tiene el atributo base. Es el tipo predefinido de datos sobre el que se construye la restricción.

```
<xs:nombre_restriccion value="" />
```

```
</xs:restriction>
```

Ejemplos de restricciones

```
<xs:element name="sexo"  
type="tipo_sexo"/>
```

```
<xs:simpleType name="tipo_sexo">  
  <xs:restriction base="xs:string">  
    <xs:enumeration value="mujer"/>  
    <xs:enumeration value="hombre"/>  
  </xs:restriction>  
</xs:simpleType>
```

```
<xs:element name="password"  
type="tipo_password"/>
```

```
<xs:simpleType name="tipo_password">  
  <xs:restriction base="xs:string">  
    <xs:pattern value="\d{3}-[A-Z]{2}"/>  
  </xs:restriction>  
</xs:simpleType>
```

```
<xs:element name="codigo_postal" type="tipo_cp"/>
```


```
<xs:simpleType name="tipo_cp">  
  <xs:restriction base="xs:string">  
    <xs:length value="5"/>  
  </xs:restriction>  
</xs:simpleType>
```

minInclusive	mínimo valor que puede tomar el número; por ejemplo, si mininclusive vale 5, el número tiene que ser mayor o igual que 5.
minExclusive	el número debe ser mayor que este valor; por ejemplo, si minExclusive vale 5, el número debe ser mayor que 5.
maxInclusive	máximo valor que puede tomar el número; por ejemplo, si maxinclusive vale 5, el número tiene que ser menor o igual que 5.
maxExclusive	el número debe ser menor que este valor; por ejemplo, si maxExclusive vale 5,el número debe ser menor que 5.
totalDigits	total de cifras en el número, incluyendo las enteras y las decimales.
fractionDigits	número de cifras decimales.
length:	número de unidades del valor literal; para la mayoría de los tipos (por ejemplo los tipos “string” y sus derivados, la faceta “length” se refiere a caracteres, pero para listas se refiere al número de elementos en la lista, y para valores binarios se refiere al número de octetos. No se puede aplicar a los tipos “integer”, “float” o “double” (para estos se puede utilizar “totalDigits”).
minLength y maxLength	valor mínimo y máximo respectivamente para la faceta “length”.
pattern	formato que debe tener el valor, especificado mediante una expresión regular tradicional.
enumeration	conjunto de posibles valores que puede tomar el dato
whiteSpace	controla la forma que tendrá el contenido de este dato una vez haya sido procesado

14. Tipo simple: simpleType - **restricción**

Elemento restriction

Puede contener las siguientes restricciones:

- **enumeration**: Se ponen los valores que puede tomar el elemento.
 - **maxExclusive, minExclusive**: Valores mínimos o máximos que puede tomar el elemento, sin incluir el último valor.
 - **maxInclusive, minInclusive**: Valores mínimos o máximos que puede tomar el elemento, incluyendo el último valor.
- 

14. Tipo simple: simpleType - restricción

Elemento restriction

Puede contener las siguientes restricciones:

- **pattern:** Expresión regular que expresa la restricción.

```
<xs:pattern value="([a-zA-Z0-9])*"/>
```

```
<xs:pattern value="\d{2}-\d{4}"/>
```

En este caso decimos que el patrón es de longitud indefinida, y que puede contener letras mayúsculas, minúsculas y números.

En este caso decimos que el patrón es de dos dígitos seguido de un guión y otros 4 dígitos. Por ejemplo, 25-6789.

14. Tipo simple: simpleType - **restricción**

Elemento restriction

Puede contener las siguientes restricciones:

- **length, maxLength, minLength:** Longitud de un elemento de tipo texto.
- **totalDigits:** Número exacto de dígitos permitidos.
- **fractionDigits:** Número máximo de decimales permitidos.



Ejemplo

Este xsd es válido para el xml que se muestra:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="diaSemana">
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="1"/>
        <xs:maxInclusive value="7"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<diaSemana xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="shiporder.xsd">7</diaSemana>
```

Ahora plantéalo para que sea válido si en vez de poner un número pusiéramos el nombre del día (por ejemplo lo valide para sábado pero no para julio)

Solución

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="diaSemana">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="lunes"/>
        ...
        <xs:enumeration value="domingo"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```


Ejercicio: edad de los trabajadores

Se desea crear un esquema que permita validar la edad de un trabajador, que debe tener un valor entero de entre 16 y 65.

Por ejemplo, este XML debería validarse:

```
<edad>28</edad>
```

Pero este no debería validarse:

```
<edad>-3</edad>
```

Ejercicio: edad de los trabajadores

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="edad"/>
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="16"/>
      <xsd:maxInclusive value="65"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Ejercicio: peso de productos

Se desea crear un esquema que permita validar un elemento peso, que puede tener un valor de entre 0 y 1000 pero aceptando valores con decimales, como por ejemplo 28.88

Ejercicio: peso de productos

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="peso" type="tipoPeso"/>
  <xsd:simpleType name="tipoPeso">
    <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="1000"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Ejercicio: pagos validados

Crear un esquema que permita validar un elemento pago en el cual puede haber cantidades enteras de entre 0 y 3000 euros.

Ejercicio: pagos validados

```
<xsd:schema
```

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
  <xsd:element name="pago" type="tipoPago"/>
```

```
  <xsd:simpleType name="tipoPago">
```

```
    <xsd:restriction base="xsd:integer">
```

```
      <xsd:minInclusive value="0"/>
```

```
      <xsd:maxInclusive value="3000"/>
```

```
    </xsd:restriction>
```

```
  </xsd:simpleType>
```

```
</xsd:schema>
```

Caso final 01- Classroom

Validar un documento XML utilizando para ello XML Schema que contiene un registro de temperaturas con la siguiente estructura:

- El elemento principal contiene un registro de temperaturas con el nombre de la provincia, la fecha de registro, la temperatura mínima y máxima y una relación de incidencias.
- La fecha de registro debe ser una fecha válida.
- La temperatura mínima no debe ser inferior a -50°C.
- La máxima no debe ser superior a 50°C.
- El número de incidencias no tiene límite.
- Las incidencias tienen un nombre que puede ser “frío”, “nieve”, “lluvia” o “calor”.
- Las incidencias tienen una severidad que puede ser “alta”, “media” o “baja”.

```
<?xml version="1.0" encoding="UTF-8"?>
<registro_temperatura>
  <provincia>Zamora</provincia>
  <fecha>2022-01-07</fecha>
  <temperatura_minima>-5</temperatura_minima>
  <temperatura_maxima>4</temperatura_maxima>
  <incidencias>
    <incidencia>
      <nombre>frio</nombre>
      <severidad>alta</severidad>
    </incidencia>
    <incidencia>
      <nombre>nieve</nombre>
      <severidad>alta</severidad>
    </incidencia>
    <incidencia>
      <nombre>lluvia</nombre>
      <severidad>media</severidad>
    </incidencia>
  </incidencias>
</registro_temperatura>
```


Caso final 02 - Classroom

Crea un validador para un documento XML con XML Schema que almacena los expedientes académicos de un conjunto de estudiantes.

La estructura del documento XML es la siguiente:

- Un documento contiene un conjunto de expedientes.
- Cada expediente contiene el nombre de la titulación, el nombre del estudiante y un conjunto de módulos por cada estudiante.
- Cada módulo tiene un atributo con el nombre, otro atributo que indica si el módulo ya ha sido aprobado y un tercer atributo que informa del curso al que pertenece el módulo y que puede tomar los valores “Primero” y “Segundo”.

```
<?xml version="1.0" encoding="UTF-8"?>
<expedientes>
  <expediente>
    <titulacion>DAW</titulacion>
    <estudiante>Ainhoa Garante Lizarraga</estudiante>
    <modulos>
      <modulo>
        <nombre>Sistemas Informáticos</nombre>
        <aprobado>true</aprobado>
        <curso>Primero</curso>
      </modulo>
      <modulo>
        <nombre>Lenguaje de marcas y sistemas de gestion de la informacion</nombre>
        <aprobado>true</aprobado>
        <curso>Primero</curso>
      </modulo>
      <modulo>
        <nombre>Programacion</nombre>
        <aprobado>false</aprobado>
        <curso>Segundo</curso>
      </modulo>
    </modulos>
  </expediente>
  <expediente>
    <titulacion>DAW</titulacion>
    <estudiante>Fernando Soler Puig</estudiante>
    <modulos>
      <modulo>
        <nombre>Programacion</nombre>
        <aprobado>true</aprobado>
        <curso>Primero</curso>
      </modulo>
      <modulo>
        <nombre>Entornos de desarrollo</nombre>
        <aprobado>false</aprobado>
        <curso>Segundo</curso>
      </modulo>
      <modulo>
        <nombre>Diseño de interfaces Web</nombre>
        <aprobado>false</aprobado>
        <curso>Segundo</curso>
      </modulo>
    </modulos>
  </expediente>
</expedientes>
```

Caso final 03 - Classroom

1. Construye una DTD que permita almacenar pedidos de clientes, con las siguientes características:
 - Los pedidos tendrán los campos: empresa o cliente que realiza el pedido (uno u otro pero no dos en el mismo pedido), productos solicitados (al menos uno), fecha, y opcionalmente un elemento llamado factura.
 - El elemento empresa tendrá en su interior un número que solo puede contener valores entre 0000 y 9999.
 - El elemento cliente tendrá en su interior un atributo obligatorio llamado NIF como identificador único y un elemento llamado MOTE de tipo textual.
 - El elemento factura se compone a su vez de los elementos emisor de tipo PCDATA, total (al menos 1) y fecha_fact de tipo PCDATA. Además tendrá dos atributos obligatorios: uno llamado numfactura que será identificador único y obligatorio y otro llamado cliente_factura, que será opcional.
1. Construye un Schema equivalente a la DTD creada.
2. Construye un XML que valide tanto tu DTD como tu XSD.



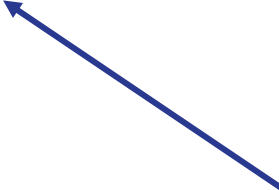
¿Y qué pasa con las etiquetas vacías que solo
tienen atributos?

15. Etiquetas vacías con atributos

```
<xs:complexType>
```

```
... atributos ...
```

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



Cuando la etiqueta solo contiene atributos, basta con indicar que es de tipo complejo y poner sus atributos dentro de la etiqueta “Complex Type”

15. Etiquetas vacías con atributos

Por ejemplo:

```
<libro titulo="El señor de los anillos" autor="John Ronald Ruelen Tolkien"  
editorial="Tirant Lo Blanch" />
```



```
<xs:element name="libro">
```

```
  <xs:complexType>
```

```
    <xs:attribute name="titulo" type="xs:string" />
```

```
    <xs:attribute name="autor" type="xs:string" />
```

```
    <xs:attribute name="editorial" type="xs:string" />
```

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

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

15. Etiquetas vacías con atributos

<xs:complexType>

```
<xs:attribute name="">
```


```
<xs:SimpleType>
```

```
...
```

```
</xs:SimpleType>
```

```
</xs:attribute>
```

</xs:complexType>



Si por el contrario el atributo tiene algún tipo de restricción, entonces el atributo se convierte en un elemento de tipo "Simple Type"

15. Etiquetas vacías con atributos

Por ejemplo:

```
<xs:element name="estacion">
```

```
<xs:complexType>
```

```
<xs:attribute name="nombre">
```

```
<xs:SimpleType>
```

```
<xs:restriction
```

```
base="xs:string">
```

```
<xs:enumeration
```

```
value="Primavera" />
```

```
<xs:enumeration value="Verano" />
```

```
<xs:enumeration value="Otoño" />
```

```
<xs:enumeration value="Invierno" />
```

```
</xs:restriction>
```

```
</xs:SimpleType>
```

```
</xs:attribute>
```

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

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

```
<estacion nombre="Primavera" />
```



Ejemplo

Dado el siguiente xml:

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

<netflix>

    <serie nombre="elite" genero="suspense" temporadas="4" anio_inicio="2018" />

    <serie      nombre="stranger      things"      genero="ficción"      temporadas="3"
estado="emisión" />

</netflix>
```

Genera su correspondiente xsd sabiendo que:

- el atributo nombre es una cadena de caracteres cualesquiera y obligatorio.
- el atributo genero es opcional y solo puede contener "suspense", "ficción", "comedia" o "variado".
- el atributo temporadas es obligatorio y solo puede contener un número y no puede ser negativo.
- el atributo anio_inicio solo puede ser un número y no puede ser menor de 0 ni mayor que el año actual. Además es optativo.
- el atributo estado es opcional y solo puede contener los valores "emisión", "finalizada" o "cancelada".