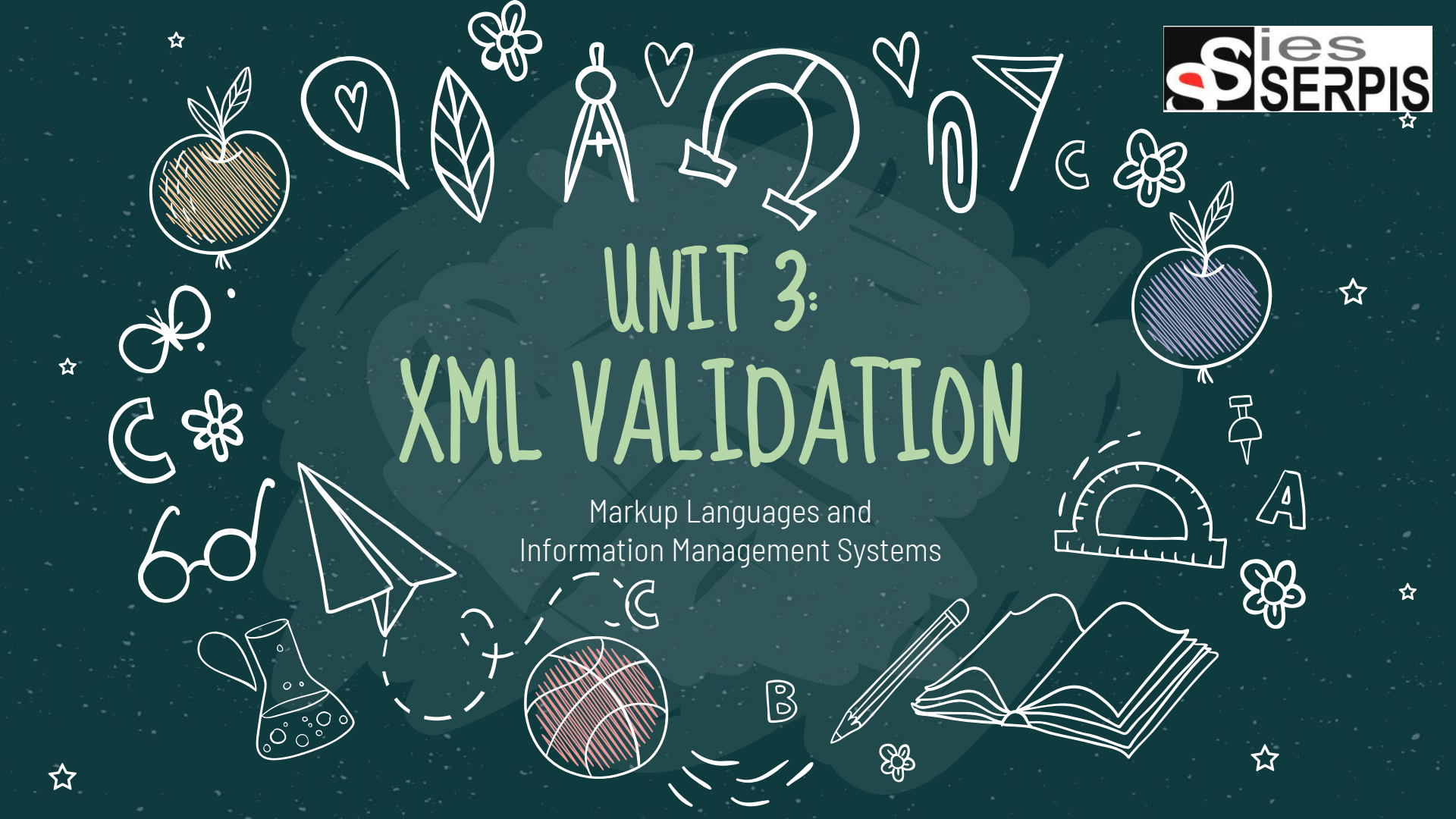

UNIT 3:
XML VALIDATION
Markup Languages and
Information Management Systems
ies SERPIS

01

DTD DECLARATION

02

DTD DEFINITION

03

NAMESPACES

04

XML SCHEMA

01. DTD DECLARATION

DTD
(Document Type Definition)

- Set of rules used to validate an XML document, specifying
  - what can be used (elements, attributes, entities and notations) to build the XML document.
  - how can it be used (order, nesting...).
- Their use is not mandatory, but it is advisable.
- They can be given both internally and externally.
- Use of the **<!DOCTYPE>** tag.

DTD DECLARATION

Internal DTD

- Definition within the same XML document.
  - In the prologue, just after the first line.
- Works both for `standalone="yes"` and `standalone="no"` (*default*).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rootName[
  <!ELEMENT rootName (childName1, childName2)>
  <!ELEMENT childName1 (#PCDATA)>
  <!ELEMENT childName2 (#PCDATA)>
]>
<rootName>
  <childName1>Text1</childName1>
  <childName2>Text2</childName2>
</rootName>
```

DTD DECLARATION

External DTD

- Definition in a separate file with `.dtd` extension
- The `<!DOCTYPE>` tag in the XML document just links to the DTD file.
- Two types of reference:
  - through a private identifier (`SYSTEM`)
  - through a public identifier (`PUBLIC`)

```
<!DOCTYPE rootName PUBLIC
  "publicIdent" "filePath">
```

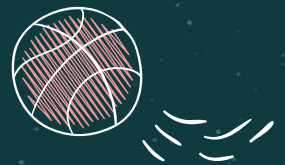
- Only works for `standalone="no"` (*default*).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rootName SYSTEM "file.dtd">
<rootName>
  <childName1>Text1</childName1>
  <childName2>Text2</childName2>
</rootName>
```

```
<!--file.dtd-->
<!ELEMENT rootName (childName1, childName2)>
<!ELEMENT childName1 (#PCDATA)>
<!ELEMENT childName2 (#PCDATA)>
```

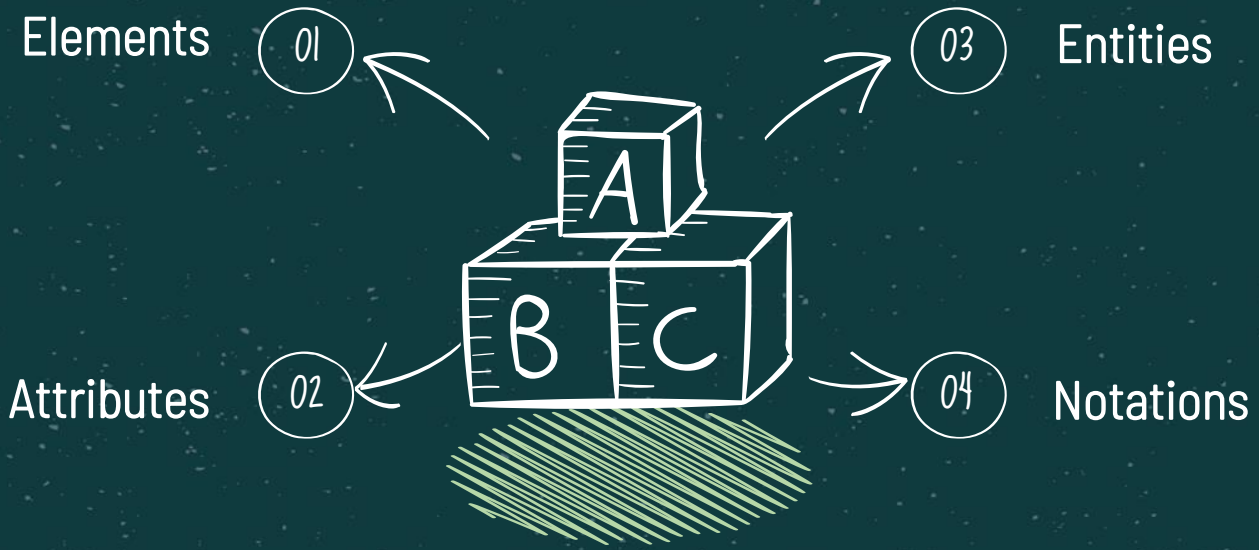
DTD DEFINITION

02.



DTD DEFINITION

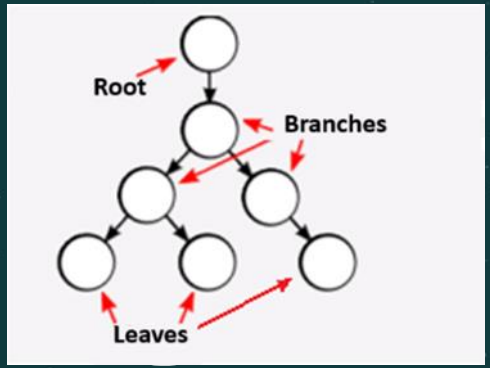
A DTD is built by defining:



DTD DEFINITION

Elements

- `<!ELEMENT>` tag
- Indicates conditions subject to the appearance of an element in the XML document.
- For the file to be valid, all elements must
  - have their type declared
  - conform to their declared type

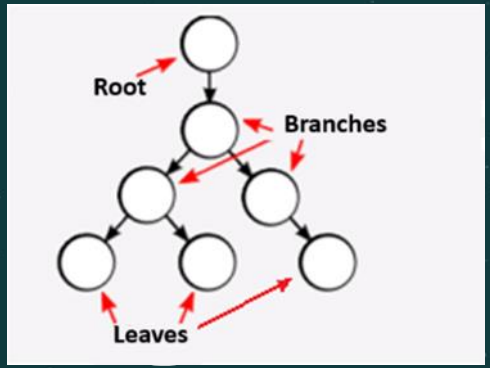


DTD DEFINITION

Elements

```
<!ELEMENT elementName contentType>
```

- Accepted content types:
  - **ANY**: element can contain anything.
    Wildcard for W.I.P. that should not be in the final version.
  - **EMPTY**: element must be empty to validate.
  - (**#PCDATA**): element contains a string.
    - `<`, `&`, `]]>` forbidden.
    - Cannot have children.
  - (*elementName*): contains another element thus named.



DTD DEFINITION

Elements

- Cardinality

NOTATION	MEANING	EXAMPLE
nothing	only one element	`<!ELEMENT notice (from)>`
?	zero or one	`<!ELEMENT notice (from?)>`
+	one or more	`<!ELEMENT notice (message+)>`
*	zero or more	`<!ELEMENT notice (message*)>`
(name1, name2, ...)	there must be all	`<!ELEMENT notice (from, to, message)>`
(name1 | name2 | ...)	there must be one	`<!ELEMENT notice (#PCDATA | to | message)*>`

DTD DEFINITION

exercise

```
<!DOCTYPE cv [
  <!ELEMENT cv (name, address,
    phone, fax?, email+, languages)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
  <!ELEMENT fax (#PCDATA)>
  <!ELEMENT email (#PCDATA)>
  <!ELEMENT languages (language*)>
  <!ELEMENT language (#PCDATA)>
]>
```

Build an XML file validated by this DTD.

DTD DEFINITION

Attributes

- `<!ATTLIST>` tag
- Indicates conditions subject to the appearance of the attributes of an element in the XML file.
- For the file to be valid, all attributes must:
  - have their type declared
  - conform to their declared type

DTD DEFINITION

Attributes

- Syntax, option 1: All attributes within the same **<!ATTLIST>** tag

```
<!ATTLIST elementName
    attribName1 attribType value
    attribName2 attribType value…>
```

- Syntax, option 2: Separated **<!ATTLIST>** tags for each attribute

```
<!ATTLIST elementName attribName1 attribType value>
<!ATTLIST elementName attribName2 attribType value>
…
```

DTD DEFINITION

Attributes

- Accepted types:

CDATA	NMTOKEN
an enumerated list	NMTOKENS
ID	ENTITY
IDREF	ENTITIES
IDREFS	NOTATION

Attributes : types

- `CDATA`: any character string (not tags).

```
<!ATTLIST person name CDATA #REQUIRED>
```

- *an enumerated list*: can only take the indicated values.

```
<!ATTLIST payment type (cash|card) "cash">
```

- `ID`: unique identifier (an element can only have one `ID` and two elements cannot have the same `ID`).

```
<!ATTLIST product code ID #REQUIRED>
```

Attributes : types

- `IDREF`: reference to the ID of another existing element.

```
<!ATTLIST employee
   idEmployee ID #REQUIRED
   idBoss IDREF #IMPLIED>
```

- `IDREFS`: like `IDREF`, but validates a list of existing, single blank space-separated `ID`.

```
<!ATTLIST employee
   idEmployee ID #REQUIRED
   idCollaborators IDREFS #IMPLIED>
```

Attributes : types

- `ENTITIY`: an entity declared within the current DTD.

  (Example in slide 25)

- `ENTITIES`: validates a list of existing, single blank space-separated `ENTITY`.

Attributes : types

- `NMTOKEN:` any valid name, without any spaces inside (ignores any before or after).

```
<!ATTLIST river birthCountry NMTOKEN #REQUIRED>
```

- `NMTOKENS:` validates a list of existing, single blank space-separated NMTOKEN.

```
<!ATTLIST river passCountries NMTOKENS #REQUIRED>
```

- `NOTATION:` notation defined in the current DTD.

(Example in slide 29)

DTD DEFINITION

Attributes

- Possible **values**:
  - *"value"*: will be the default value of the attribute.
  - `#REQUIRED`: mandatory attribute.
  - `#IMPLIED`: optional attribute.
  - `#FIXED` *"value"*: the attribute will always have the same value.

```
<!ATTLIST message sender CDATA #FIXED "IES Serpis">
```

DTD DEFINITION

example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cinema [
  <!ELEMENT cinema (films, directors)>
  <!ELEMENT films (film)*>
  <!ELEMENT film (#PCDATA)>
  <!ATTLIST film filmcode ID #REQUIRED>
  <!ELEMENT directors (director)*>
  <!ELEMENT director (#PCDATA)>
  <!ATTLIST director filmography IDREFS #REQUIRED>
]>
<cinema>
  <films>
    <film filmcode="F1">Aliens</film>
    <film filmcode="F2">Gran Torino</film>
    <film filmcode="F3">The Terminator</film>
    <film filmcode="F4">Titanic</film>
  </films>
  <directors>
    <director filmography="F2">Clint Eastwood</director>
    <director filmography="F1 F3 F4">James Cameron</director>
  </directors>
</cinema>
```

Cinema XML file validated by internal DTD

DTD DEFINITION

Exercise: cooking recipe

Write the DTD that validates an XML document containing a cooking recipe with the following structure:

- The main element is **recipes**, which is a collection of **recipe**.
- A recipe consists of its **name** (free text), a list with one or more **ingredients** and a text with the **preparation** instructions (free text).
- An **ingredient** contains free text.
- Recipes have an attribute telling the number of **servings**.
- Recipes have an attribute to indicate **difficulty** (high, medium or low; default is medium).
- Ingredients have an optional attribute with the **quantity** of the ingredient.
- Ingredients have an optional attribute to indicate the measuring **unit** in which the quantity is expressed.

DTD DEFINITION

Entities

- `<!ENTITY>` tag.
- Like constants in programming:
  - they are given a name.
  - they are assigned an invariable value.
- Example: internal character entity

  `<!ENTITY copy "&#169;">`

  ©
  - Can be summoned later as `&copy;`
    - exception as an attribute value: `attribName="copy"`
  - Can be generalized to any string.

DTD DEFINITION

Entities

- Can be internal (within XML document) or external.
- External entities can be:
  - public (`PUBLIC`) or private (`SYSTEM`).
  - parsed or not parsed.
    - parsed: linked file contains plain text.
    - not parsed: skipped by the XML parser; parsing done by final application.
      - Uses **NDATA** and notation.

DTD DEFINITION

entities: examples

```
<!--internal-->
<!ENTITY insti "IES Serpis">

<!--external private parsed-->
<!ENTITY country SYSTEM "country.xml">

<!--external private not parsed->
<!DOCTYPE image [
  <!ELEMENT image EMPTY>
  <!ATTLIST image source ENTITY #REQUIRED>
  <!ENTITY logo SYSTEM "logo.gif" NDATA gif>
  <!NOTATION gif SYSTEM "image/gif">
]>
<image source="logo"/>
```

DTD DEFINITION

entities: examples

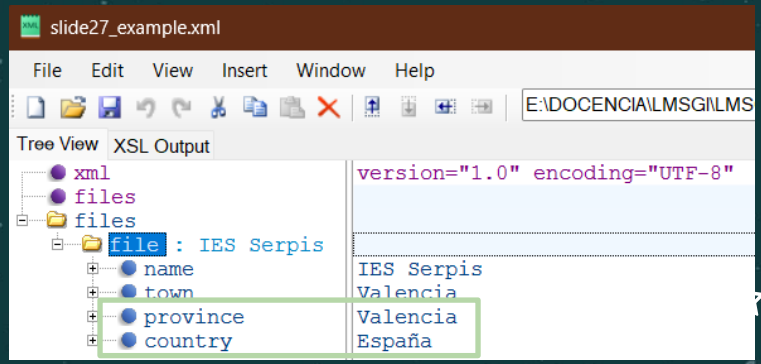
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE files [
  <!ELEMENT files (file)>
  <!ELEMENT file (name, town, province, country)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT town (#PCDATA)>
  <!ELEMENT province (#PCDATA)>
  <!ELEMENT country (#PCDATA)>
  <!ENTITY townName "Valencia">
  <!ENTITY provinceName SYSTEM "prov.txt">
  <!ENTITY country SYSTEM "country.xml">
  <!ENTITY insti "IES Serpis">
]>
<files>
  <file>
    <name>&insti;</name>
    <town>&townName;</town>
    <province>&provinceName;</province>
    &country;
  </file>
</files>
```

DTD DEFINITION

Entities

- Most applications block external entities because they pose a security risk (*XXE injection* attack).
- For this reason, web browsers block and do not display them.
- A non-blocking application must be used to check that it works, such as XML Notepad.

```
−<files>
  −<file>
      <name>IES Serpis</name>
      <town>Valencia</town>
      <province/>
    </file>
</files>
```



DTD DEFINITION

Notations

- Identify the format of non-XML entities (also attributes).
  - often, MIME types.
  - public (**PUBLIC**) or private (**SYSTEM**).

```
<!NOTATION gif SYSTEM "image/gif">
```

DTD DEFINITION

notations: example with attribute

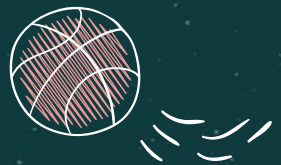
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE animals [
  <!ELEMENT animals (animal)*>
  <!ELEMENT animal (name)>
  <!ELEMENT name (#PCDATA)>
  <!NOTATION png SYSTEM "image/png">
  <!ATTLIST animal
      image CDATA #IMPLIED
      imgType NOTATION (png) #IMPLIED> ]>
<animals>
  <animal image="dog.png" imgType="png">
    <name>Dog</name>
  </animal>
</animals>
```

Check slide 25 for an example on entities.

PRÁCTICA 3.1

NAMESPACES

03.



NAMESPACES

- A namespace is a set of unique names.
- They arise from the need to differentiate distinct elements with the same name when code coming from several files is combined into one.

```
<root>
  <pen>
    <material>plastic</material>
    <brand>BIC</brand>
    <ink_color>blue</ink_color>
  </pen>
  <pen>
    <animals>
      <rooster>Cocky</rooster>
      <hen>Sugar</hen>
      <hen>Candy</hen>
    </animals>
  </pen>
</root>
```

Distinct elements with the same name

NAMESPACES

- Declared using the `xmlns` attribute.
  - Value: the namespace identifier (actually, a URL to its definition).
- It can be done in two ways:
  1) *inline* (directly in the element).
  2) in the root node of the document, using prefixes.

```
<root>
  <pen xmlns="URL1">
    <material>plastic</material>
    <brand>BIC</brand>
    <ink_color>blue</ink_color>
  </pen>
  <pen xmlns="URL2">
    <animals>
      <rooster>Cocky</rooster>
      <hen>Sugar</hen>
      <hen>Candy</hen>
    </animals>
  </pen>
</root>
```

NAMESPACES

- Declared using the `xmlns` attribute.
  - Value: the namespace identifier (actually, a URL to its definition).
- It can be done in two ways:
  1) *inline* (directly in the element).
  2) in the root node of the document, using prefixes.
     - The prefix must be used with the element you need to differentiate and all its children.
     - Also, in their closing tags.

2

```
<root xmlns:pr1="URL1" xmlns:pr2="URL2">
  <pr1:pen>
    <pr1:material>plastic</pr1:material>
    <pr1:brand>BIC</pr1:brand>
    <pr1:ink_color>blue</pr1:ink_color>
  </pr1:pen>
  <pr2:pen>
    <pr2:animals>
      <pr2:rooster>Cocky</pr2:rooster>
      <pr2:hen>Sugar</pr2:hen>
      <pr2:hen>Candy</pr2:hen>
    </pr2:animals>
  </pr2:pen>
</root>
```

04. XML SCHEMA

XML SCHEMA

- Defines types
- Allows to validate XML

Compared to DTD, is:
- More powerful and accurate
- More current
- More complex

Recommendation

XML Schema Definition (XSD)

Written using XML syntax
- Can be checked if well-formed.
- Can be manipulated via XML DOM.
- Can be transformed using XSLT.

.xsd files
- Always external

XML Schema

More powerful

In addition to everything that DTD allows, XML Schema...

- allows to define data types (*integer*, *date*...).
  - Even create more.
- Allows to add restrictions:
  - Number ranges (maximum, minimum).
  - Patterns for strings.
  - More accurate cardinality.

XML SCHEMA

Please pay attention to the color code used in the examples to differentiate between:

- **XML file**
- **XSD file**

Structure of the XSD document

- First line: the same as any other XML document.
- Root node: always `<somePrefix:schema>`
  - Has attributes; at least, one defining the namespace:

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

  - This namespace is always the same.
  - All of the XML Schema elements and data types are defined there.
- The remaining elements and attributes are declared between the root node opening and closing tags.

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


  <!--Remaining elements and attributes-->



</xs:schema>
```

XML SCHEMA

XML Schema instance

- This is what validated XML documents are called in this context.
- The link with the XSD must be done within the root node opening tag.
- For XSD in a **local** file:

```
<rootNode xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="path_to_xsd_file">
```

  - This namespace is always the same.
  - It contains all the attributes needed for the link.

```
<?xml version="1.0" encoding="UTF-8"?>
<message
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="xsd/message.xsd">
  <from>Luisa</from>
  <to>Carla</to>
  <body>I'm on my way</body>
</message>
```

XML SCHEMA

example

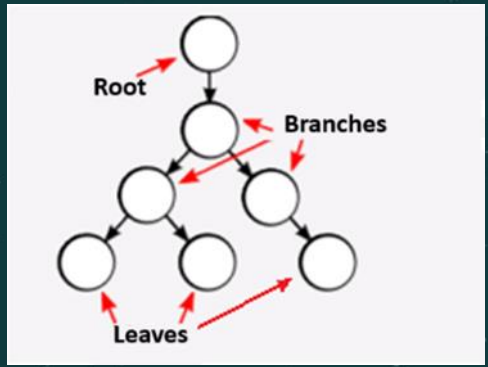
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="message">
    <xs:complexType>
      <xs:sequence>

        <xs:element name="from" type="xs:string"/>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>

      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

The xs: prefix was used in all XSD code examples

XML SCHEMA

Elements

- `<xs:element>` tag.
- Simple elements: can only contain "text".
  - neither other elements nor attributes.
- Complex elements : do contain other elements and/or attributes.
  - Can be:
    - empty (with attributes).
    - container only for other elements.
    - container only for text (with attributes).
    - mixed (container for text and other elements).

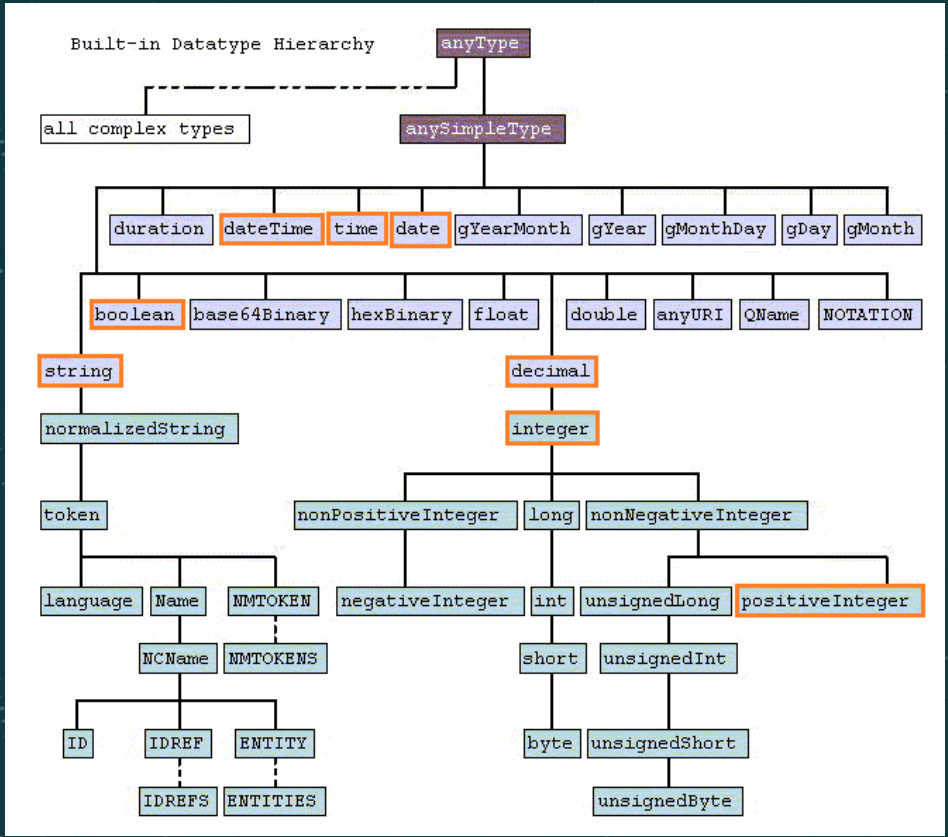


Simple elements

- Only contain "text".
- The data type of this text must be specified.
- Syntax:

```
<xs:element name="nodeName" type="xs:dataType"/>
```

XML SCHEMA



The most common data type are outlined in orange.

Simple elements

- Other attributes of `<xs:element>` :
  - `default="value"`: indicates a default value.
  - `fixed="value"`: indicates a fixed value.
  - `minOccurs, maxOccurs`: minimum and maximum number of occurrences.
    - default "**1**".
    - unlimited with "**unbounded**".
  - `ref`: reference to a global element (defined separately)
    - incompatible with **name**, **type** and `<xs:simpleType>`

XML SCHEMA

simple elements

```
<!--instance fragment-->
<surname>Bru</surname>
<age>17</age>
<birthDate>2007-12-05</birthDate>
```

```
<!—schema fragment-->
<xs:element name="surname" type="xs:string"/>
<xs:element name="age" type="xs:positiveInteger"/>
<xs:element name="birthDate" type="xs:date"/>
```

Attributes

Only simple types!

- `<xs:attribute>` tag.
- Have no order.
- Syntax:

  `<xs:attribute name="attributeName" type="xs:dataType"/>`
- Other attributes of `<xs:attribute>`:
  - `default`
  - `fixed`
  - `use`: obligation. `required|optional` (default)
  - `ref`: reference to global attribute (defined separately)
    - incompatible with `name`, `type` and `<xs:simpleType>`

```
<xs:attribute name="color" type="xs:string" default="blue" use="required"/>
```

Restrictions

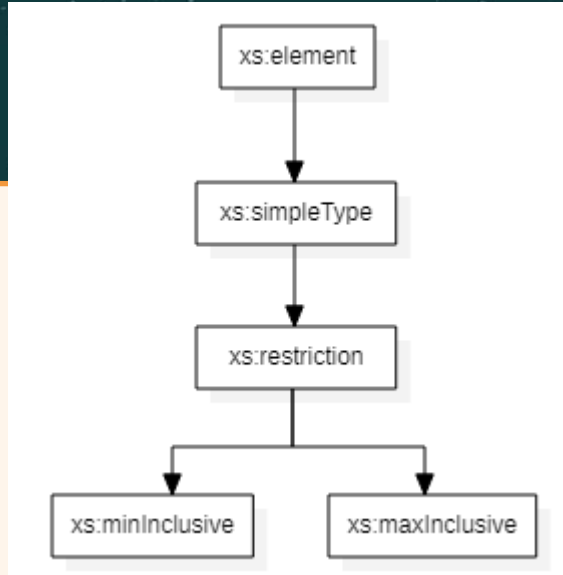
- A.K.A. "facets".
- Define acceptable values for elements and attributes.
    - Reduce the original domain.
- To use them, several auxiliary tags must be entered.

XML SCHEMA

restrictions: example 1

Data type not here!

```
<!--schema fragment-->
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```



Element with a limited age between 0 and 120 (inclusive).

The restriction reduces the domain of the given basic type.

XML SCHEMA

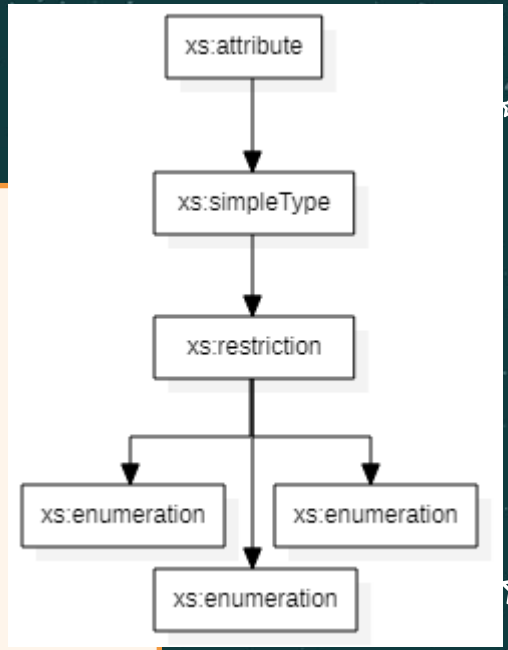
Restrictions

`minInclusive`	Value must be greater than or equals to this value
`maxInclusive`	Value must be lower than or equals to this value
`minExclusive`	Value must be greater than this value
`maxExclusive`	Value must be lower than this value
`totalDigits`	Maximum number of digits
`fractionDigits`	Maximum number of decimal places
`length`	Exact length of a string or list
`minLength`	Minimum length of a string or list
`maxLength`	Maximum length of a string or list
`enumeration`	Accepted value (in an enumerated list)
`pattern`	String pattern (RegEx)

XML SCHEMA

restrictions: example 2

```
<!--schema fragment-->
<xs:attribute name="color">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="red"/>
      <xs:enumeration value="amber"/>
      <xs:enumeration value="green"/>
   </xs:restriction>
  </xs:simpleType>
</xs:element>
```



Attribute that only can have values from an enumerated list (colors on a traffic light).

The restriction reduces the domain of the given basic type.

XML SCHEMA

Local vs. global declaration

Two ways to declare elements and attributes:

a) LOCAL: directly in the element or attribute (*inline*)

b) GLOBAL: with reference

- ...to an existing element or attribute, using the `ref` attribute
- ...to an existing `<xs:complexType>`, using the `type` attribute
  - used to refer complex elements (*example in slide 58*).
- Makes reading and reutilization easier (the latter, only complex types).

XML SCHEMA

local declaration

```
<!--instance fragment->
<vegetable code="A9">potato</vegetable>
```

```
<!--schema fragment->
<xs:element name="vegetable" minOccurs="0"
    maxOccurs="unbounded">
  <xs:complexType mixed="true">
    <xs:attribute name="code">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:pattern value="[A-Z][0-9]"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

XML SCHEMA

global declaration with ref

```
<!--instance fragment->
<vegetable code="A9">potato</vegetable>
```

```
<!--schema fragment-->
<xs:element name="vegetable">
  <xs:complexType mixed="true">
    <xs:attribute ref="code"/>
  </xs:complexType>
</xs:element>

<xs:attribute name="code">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

No further attributes!

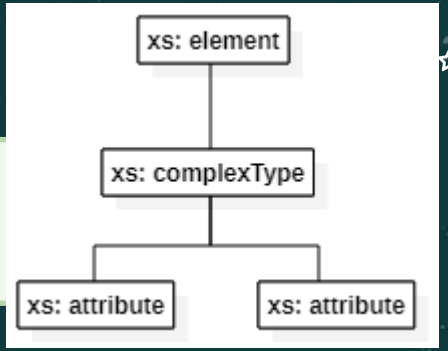
Complex elements

- Contain other elements and/or attributes.
- Can be:
  - empty (with attributes).
  - container only for other elements.
  - container only for text (with attributes).
  - mixed (text + other elements).
- Content defined within an `<xs:complexType>` element.

XML SCHEMA

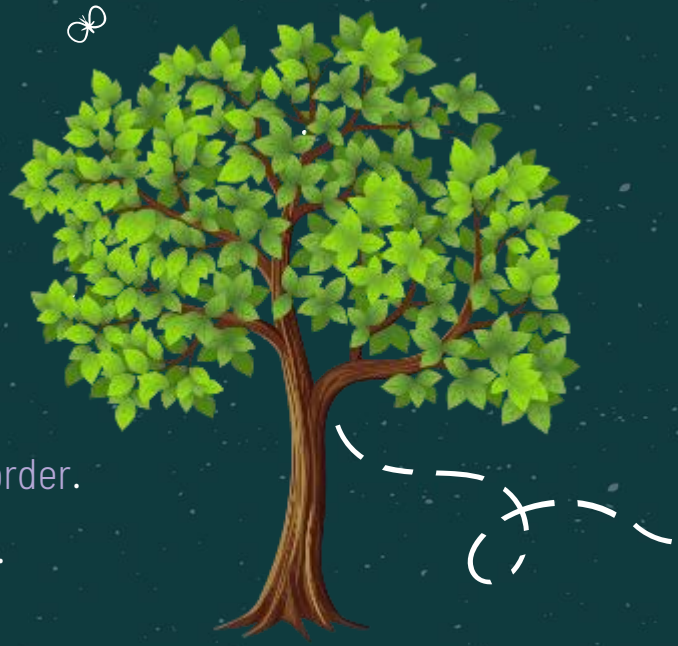
empty complex element

```
<!--instance fragment-->
<vegetable code="A9"/>
```




```
<!--schema fragment-->
<xs:element name="vegetable">
  <xs:complexType>
    <xs:attribute name="code" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

Complex elements



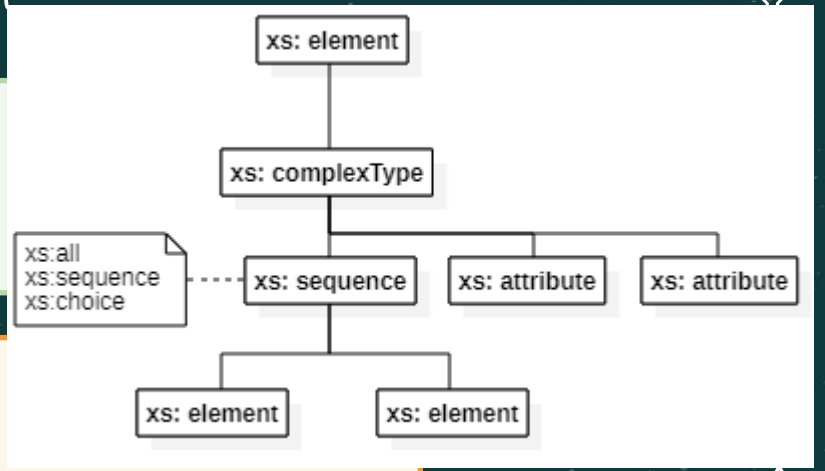
Containers only for other elements:

- These elements are children of one of these three nodes:
  - `<xs:choice>`: only one of the elements must appear.
  - `<xs:all>`: all the elements must appear, no matter the order.
  - `<xs:sequence>`: all the elements must appear, in order.
- The root node of the instance is of this type.

XML SCHEMA

container element of elements

```
<!--instance fragment-->
<professor glasses="true">
  <name>Guillermo</name>
  <surname>Domingo</surname>
</professor>
```

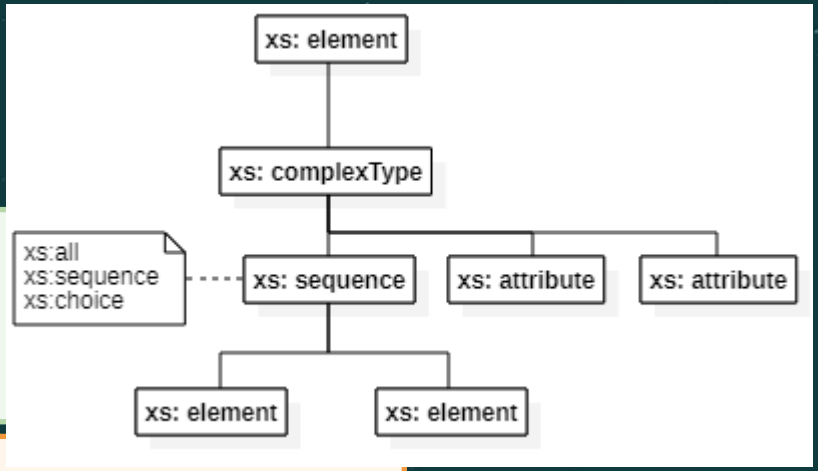



```
<!--schema fragment (LOCAL)-->
<xs:element name="professor">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="surname" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="glasses" type="xs:boolean"/>
  </xs:complexType>
</xs:element>
```

XML SCHEMA

container element of elements

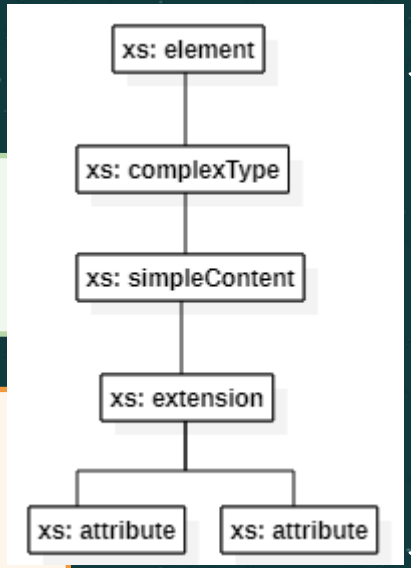
```
<!--instance fragment-->
<professor glasses="true">
  <name>Guillermo</name>
  <surname>Domingo</surname>
</professor>
```




```
<!--schema fragment (GLOBAL)-->
<xs:element name="professor" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="gardener" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="surname" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="glasses" type="xs:boolean"/>
</xs:complexType>
```

XML SCHEMA
complex element containing only text

```
<!--instance fragment-->
<shoe_size country="Spain">43</shoe_size>
```




```
<!--schema fragment (LOCAL)-->
<xs:element name="shoe_size">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:positiveInteger">
        <xs:attribute name="country" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

The amount of auxiliary tags required makes it convenient to use global declaration to improve readability.

And the attribute has no restrictions...

Complex elements

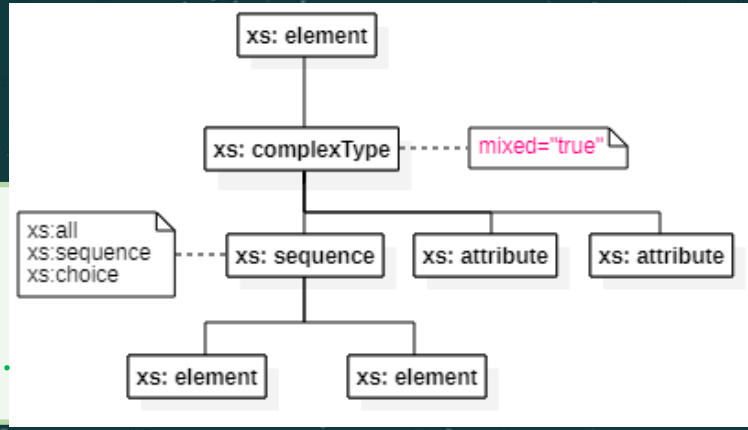
Mixed complex elements

- Are elements that contain text and other elements and attributes.
- When defining them, one needs to indicate the `mixed="true"` attribute within the `<xs:complexType>` tag.

XML SCHEMA

mixed complex element

```
<!--instance fragment-->
<letter>
  Dear Mr. <name>Pepe Pérez</name>.
  Your order number <orderNo>1032</orderNo>
  will ship on <shipDate>2024-10-31</shipDate>.
</letter>
```




```
<!--schema fragment (LOCAL)-->
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderNo"
          type="xs:positiveInteger"/>
      <xs:element name="shipDate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

EXTENSION

- Simple and complex types declared globally can be extended (by adding elements and attributes).
- This is what the `<xs:extension>` tag is for.
- The parent of this tag is:
  - `<xs:simpleContent>` when a simple type is extending.
  - `<xs:complexContent>` when a complex type is extending.

XML SCHEMA

complex type extension

In this example, the complex "personinfo" type (*from slide 58*) is extended.

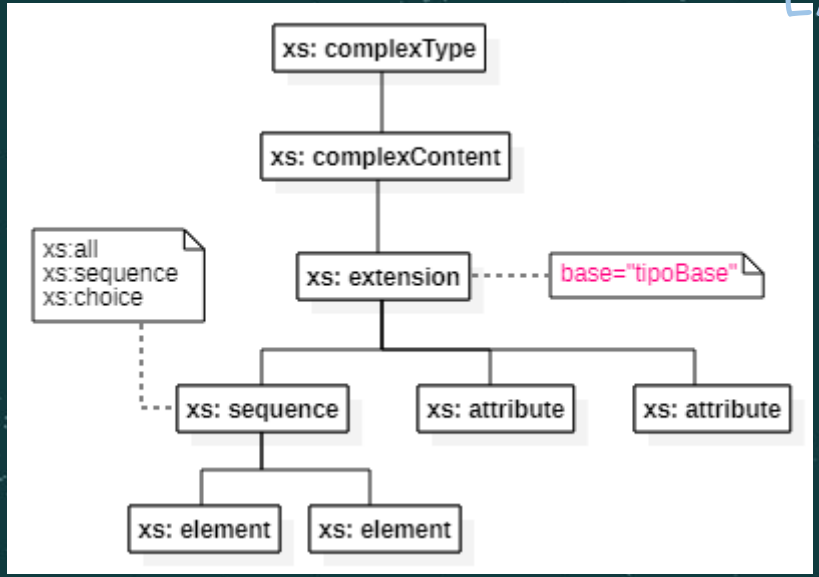
```
<!--instance fragment-->
<student glasses="true" contAssessment="false">
  <name>Absentista</name>
  <surname>Fernández</surname>
  <address>C/ Joncito, 2</address>
  <phone>612345678</phone>
</student>
```

```
<!--schema fragment (GLOBAL)-->
<xs:element name="student" type="personInfoExt"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="surname" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="glasses" type="xs:boolean"/>
</xs:complexType>
```

```
<!--continued from the code on the left-->
<xs:complexType name="personInfoExt">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address"
            type="xs:string"/>
        <xs:element name="phone"
            type="xs:positiveInteger"/>
      </xs:sequence>
      <xs:attribute name="contAssessment"
          type="xs:boolean"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XML SCHEMA

EXTENSION



The parent of `<xs:extension>` here is `<xs:complexContent>` because `personinfo` (the base for this extension) is a complex type.

XML SCHEMA

Exercise: cooking recipe again

Write the XSD that validates an XML document containing a cooking recipe with the following structure. Use the appropriate data types:

- The main element is **recipes**, which is a collection of **recipe**.
- A recipe consists of its **name** (free text), a list with one or more **ingredients** and a text with the **preparation** instructions (free text).
- An **ingredient** contains free text.
- Recipes have an attribute telling the number of **servings**.
- Recipes have an attribute to indicate **difficulty** (high, medium or low; default is medium).
- Ingredients have an optional attribute with the **quantity** of the ingredient.
- Ingredients have an optional attribute to indicate the measuring **unit** in which the quantity is expressed.

PRÁCTICA 3.2

THANKS!

Do you have any questions?

g.domingomartinez@edu.gva.com



CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**

Please keep this slide for attribution.