GESTIONE DEI DOCUMENTI IN FORMATO XML

XML (eXtensible Markup Language) nasce come linguaggio **per la definizione di linguaggi di markup** (nasce da SGML Standard Generalized Markup Language). I linguaggi, come xml, che consentono di definire altri linguaggi vengono definiti **metalinguaggi**.

XML infatti è un linguaggio standard (realizzato da W3C, World Wide Web Consortium, un ente non governativo che h lo scopo di sviluppoare le potenzialità del web) che consente di definire nuovi tag. La sua nascita è dovuta al fatto che negli anni '90 i principali browser (Microsoft e Netscape) definivano, ciascuno, dei propri tag per il linguaggio HTML (questo periodo è noto in informatica come periodo delle "guerra dei browser"), l'ente di standardizzazione W3C era quindi costretto ad "inseguire" i nuovi standard *de facto* che continuavano ad essere prodotti ad ogni versione dei diversi browser.

Ad un certo punto (anni '90) W3C, per esigenze di standardizzazione fu costretta a decidere cosa faceva parte di HTML e cosa no. Poiché comunque diversi produttori software manifestavano la necessità di creare linguaggi con aggiunta di nuovi tag rispetti "ufficiali", W3C ideò XML**, un linguaggio standardizzato che consente la creazione di nuovi tag.**

XML è quindi un linguaggio che consente di creare nuovi linguaggi di markup. Ne sono un esempio CML (Chemical Markup Language), EPUB (Electronic Pubblication, un linguaggio per la realizzazione di ebook) e SVG (Scalable Vector Graphics, un linguaggio per la rappresentazione di immagini vettoriali bidimensionali vettoriali), FXML (FX Markup Language, linguaggio per realizzare interfacce grafiche in Java)

Poiché XML, grazie alla modalità gerarchica con cui vengono utilizzati i tag nei linguaggi di markup, consente di descrivere documenti strutturati, nel tempo si è imposto anche con un'altra funzione (oltre a quella di creare nuovi linguaggi), ossia è diventato uno standard per la memorizzazione e l'interscambio scambio di dati fra applicazioni software diverse e DBMS di produttori diversi.

A differenza di HTML, il cui scopo è quello di indicare delle regole di formattazione del testo interpretate dai browser, **XML consente di definire dei tag per rappresentare dei documenti.**

Come in HTML, infatti, anche in XML i tag hanno una struttura gerarchica (un elemento dentro l'altro) e questo consente di rappresentare i documenti secondo una struttura ad albero.

I documenti XML sono file di testo salvati con estensione .xml

SOSTANZIALMENTE XML non fa niente, XML **descrive** la struttura di un dato. Tale dato potrà poi essere interpretato da un software, scritto in un qualsiasi linguaggio (Java, PHP ecc.), che utilizzerà il contenuto di quel documento XML, analogamente a quanto avviene per i file CSV.

L'operazione di "interpretare i dati di un file XML" da parte di un software è detta **parsing**.

Esempio p. 93: Esempio di documento xml che rappresenta due utenti di un sistema informatico:

ESEMPIO

Il seguente file di testo è un documento XML che rappresenta gli indirizzi di posta elettronica e le *password* degli utenti di un sistema informatico:

```
<?xml version="1.0" ?>
<users>
  <!-- Nota: le password sono cifrate. -->
  <user>
    <username>Pippo</username>
    <email>pippo32@disney.com</email>
    <password>0A1B2C3D4E5F</password>
  </user>
  <user>
    <username>Pluto</username>
    <email>pluto30@disney.com</email>
    <password>A9B8C7D6E5F4</password>
  </user>
</users>
```

Rappresentazione ad albero dello stesso documento

ESEMPIO

Il documento dell'esempio precedente presenta la struttura ad albero di FIGURA **1**, che deriva dalla struttura di nidificazione dei *tag* con l'elemento che racchiude tutti gli altri e che diviene il nodo radice dell'albero.

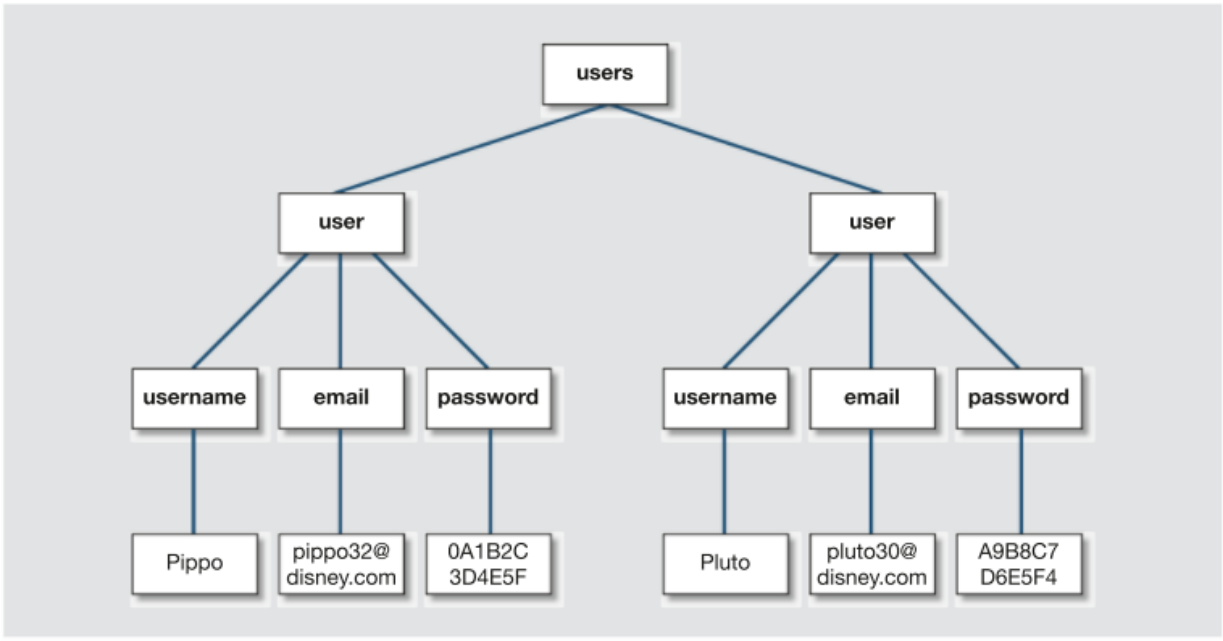


FIGURA **1**

La X di XML siginfica Extensible, estendibile. Perchè?

Perché, ad esempio, ipotizzando che il documento xml sopra descritto venga interpretato da un apposito software (chiamiamolo software A) che ne estrae le informazioni (userneme, email ecc…),

nel caso in cui il documento XML venisse ESTESO con l'aggiunta di nuove informazioni (ad esempio <indirizzo>Via Marconi 12 Milano</indirizzo> per ogni elemento "user"), il "software A" rimarrebbe comunque in grado, anche successivamente alla modifica, di leggere il documento e trovare le informazioni (userneme, email,…ad esclusione di quelle nuove naturalmente). Ovviamente affinché il "software A" possa trovare anche le informazioni nuove (ad esempio l'indirizzo) dovrà essere modificato.

Le regole di base della sintassi di XML

- Tutti i documenti devono iniziare con il prologo

  <?xml version="1.0" ?>

  Il prologo identifica la versione di XML a cui si riferisce il documento.

  Se il file che contiene un documento XML contiene caratteri che non appartengono alla codifica ASCII, si può specificare la codifica utilizzata. La codifica va specificata **nella dichiarazione iniziale** del documento.

  Ad esempio:

  Codifica che comprende i primi caratteri Unicode (i più utilizzati)

  <? xml version="1.0" encoding="ISO-8859-1" ?>

  (altre codifiche UTF-8 (da 8 a 32 bit) , UTF-16 (sempre 16 bit, quella usata da Java per i caratteri) la codifica predefinita è UTF-8)

- Ogni tag ha un nome. I nomi dei tag possono iniziare solamente con una lettera o con il carattere underscore "_".

- I nomi non possono contenere la parola xml o XML.

- **I nomi dei tag sono case sensitive**

- I nomi dei tag possono contenere esclusivamente caratteri alfabetici, cifre numeriche e i simbloli "_" , "-" e "." (underscore, meno e punto, quindi escluso lo spazio).

- Tutti i tag devono essere chiusi con un tag corrispondente avente lo stesso nome e preceduto da "/" (slash).

- La nidificazione dei tag dev'essere rigorosamente gerarchica

- L'intero documento XML, ad esclusione della dichiarazione iniziale, deve essere compreso fra i tag di apertura e di chiusura di un elemento chiamato **elemento radice.**

- I tag possono avere degli attributi. Per questi attributi è possibile specificare di valori, tali valori devono essere sempre compresi fra apici ('valore') o virgolette ("valore")

- I commenti si inseriscono con:

  <!-- *commento* -->

- I caratteri speciali, mostrati di seguito, che renderebbero il documento **mal formato** (vedremo dopo cosa significa), vanno sostituiti con le rispettive entità XML di seguito rappresentate

Carattere	Entità
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&apos;

- XML, diversamente da HTML, conserva gli spazi bianchi all'interno dei suoi documenti

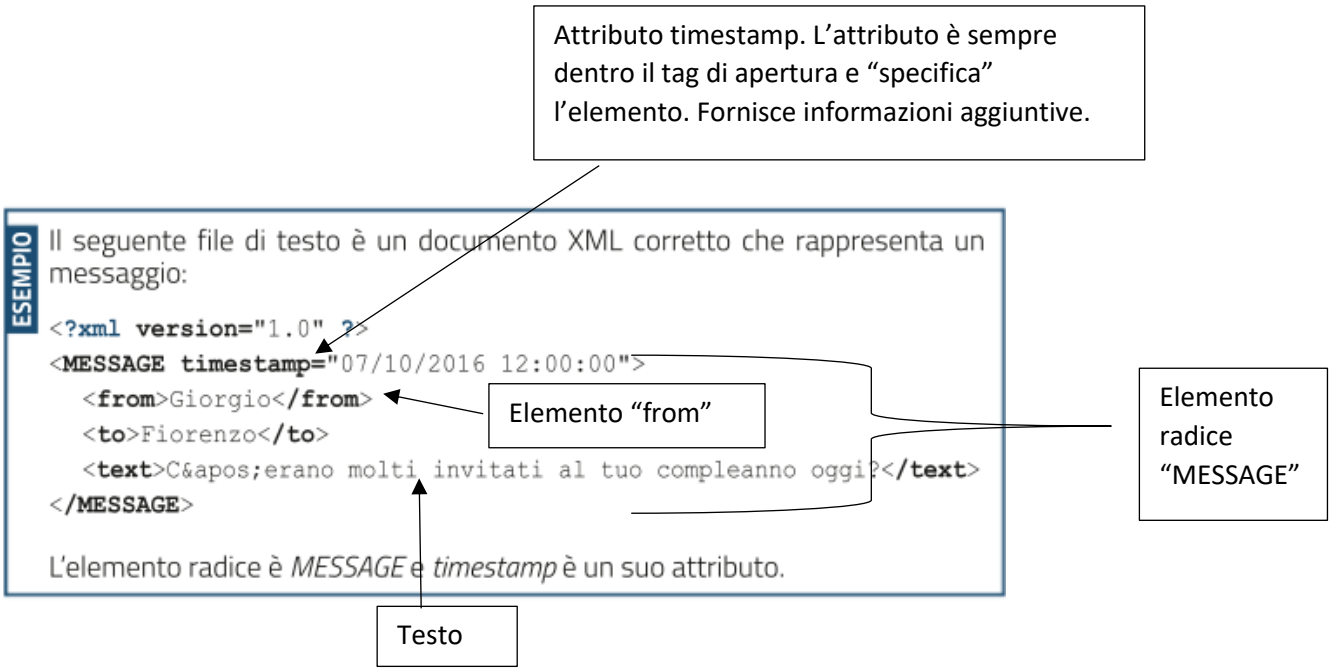
Definizione di **elemento** XML:

Un **elemento** XML è tutto ciò che è compreso fra un tag di apertura e un tag di chiusura **(INCLUSI!).**
Un elemento può contenere:

- del **testo**
- degli **attributi**
- altri **elementi**.

oppure può essere **vuoto.**

esempio a p. 95 in cui sono mostrati i vari elementi.



Differenza fra attributi ed elementi

Ma che differenza c'è fra attributi ed elementi? Quando uso uno e quando l'altro? Non c'è una regola precisa, o meglio, ci sarebbe, ma in alcuni casi non è proprio seguitissima. La regola è che i **DATI** vanno negli elementi, i **METADATI** (dati che "spiegano i dati") vanno negli attributi. Nell'esempio mostrato l'attributo timestamp indica quando è stato prodotto il messaggio, in questo caso viene considerato un attributo (descrive il dato), ma in altri casi potrebbe essere considerato un elemento e quindi inserito all'interno dell'elemento MESSAGE.

Nei documenti XML creati da noi utilizzeremo la seguente Regola empirica: limitare l'utilizzo degli attributi perché non consentono di creare struttura ad albero (non possono esistere attributi con dentro attributi) e non sono facilmente espandibili.

Sintassi speciale

1. Sintassi abbreviata per gli elementi vuoti o che comprendono solo attributi. Tale sintassi impiega un solo tag che apre e chiude l'elemento

   <tag/> tag vuoto

   <tag timestamp="10/10/2018" /> tag solamente con attributi

2. **Attributo predefinito xml:lang** che indica la lingua del testo contenuto in un elemento, si utilizza in un elemento che contiene del testo, ad esempio il seguente elemento text:

Esempio p. 96

ESEMPIO

Il documento XML dell'esempio precedente dovrebbe riportare l'attributo per l'indicazione del linguaggio:

```
<?xml version="1.0" ?>
<MESSAGE timestamp="07/10/2016 12:00:00">
  <from>Giorgio</from>
  <to>Fiorenzo</to>
  <text xml:lang="it">
    C&apos;erano molti invitati al tuo compleanno oggi?
  </text>
</MESSAGE>
```

3. **CDATA (Character DATA).**

Abbiamo detto che un elemento XML può contenere altri elementi XML. I software che gestiscono i documenti XML analizzano l'intero contenuto del documento per individuare gli elementi. Se si vuole che il contenuto di un elemento **NON venga analizzato**, per consentire di inserire qualunque sequenza di caratteri (ad esempio del codice HTML o XML), si deve qualificare tale contenuto come Character DATA utilizzando l'apposito tag:

<![CDATA[

testo…. ad esempio altro codice XML

]]>

Esempio p. 96

ESEMPIO

Il seguente file di testo è un documento XML corretto che rappresenta un messaggio:

```
<?xml version="1.0" ?>
<MESSAGE timestamp="07/10/2016 12:00:00">
  <from>Giorgio</from>
  <to>Fiorenzo</to>
  <text>
    <![CDATA[
      Ti invio un esempio di XML:
      <?xml version="1.0" ?>
      <users>
        <user>
          <username>Pippo</username>
        <email>pippo32@disney.com</email>
        </user>
        <user>
          <username>Pluto</username>
          <email>pluto30@disney.com</email>
        </user>
      </users>
    ]]>
  </text>
</MESSAGE>
```

La struttura di documento XML corretto si può sempre rappresentare con una struttura ad albero dove il nodo radice è l'elemento radice dell'albero e gli altri nodi sono:

- altri elementi
- attributi degli elementi
- contenuti testuali degli elementi
- eventuali commenti

Esempio p.146

ESEMPIO

Il seguente file di testo è un documento XML corretto che rappresenta un libro:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<libro genere="fantascienza" xml:lang="it">
   <autore>Stefano Benni</autore>
   <titolo>Terra!</titolo>
   <editore>Feltrinelli</editore>
   <anno>1983</anno>
</libro>
```

Esso è rappresentato dal diagramma ad albero di FIGURA **2**.

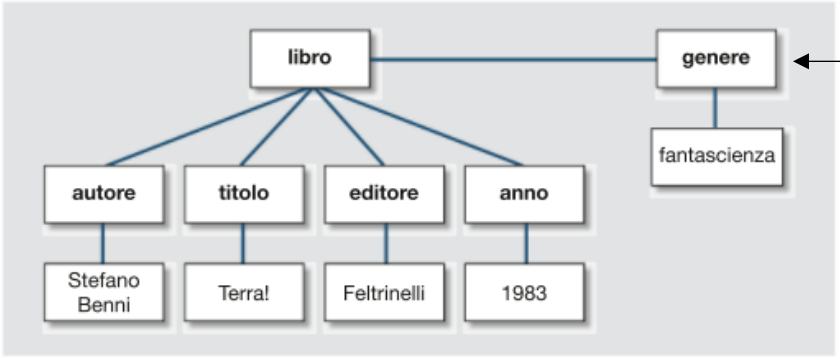


FIGURA **2**

Si noti che l'attributo viene rappresentato allo stesso livello gerarchico dell'elemento che descrive e non come "figlio"

LA DEFINIZIONE DI LINGUAGGI XML MEDIANTE SCHEMI XMD (cap 2)

Un documento XML si dice **ben formato** quando rispetta le regole sintattiche (chiusura di tutti i tag aperti, gerarchia dei tag ecc..).

Per verificare se un documento XML è ben formato (e anche se è valido, vedremo dopo cosa significa) vi sono diversi siti online, ad esempio quello al seguente link: http://www.utilities-online.info/xsdvalidation/#.W_LljuhKjIU. Per capire come effettuare la validazione è necessario spiegare cosa è uno schema XSD.

Un documento potrebbe essere ben formato ma impiegare i tag previsti in modo non corretto, come nel seguente esempio:

Esempio p. 99

ESEMPIO

Il seguente documento XML che rappresenta una collezione di libri è ben formato:

```
<?xml version="1.0" ?>
<libri>
  <libro genere="fantascienza">
    <autore>Stefano Benni</autore>
    <titolo>Terra!</titolo>
    <editore>Feltrinelli</editore>
    <anno>83</anno>
  </libro>
  <libro genere="romanzo">
    <scrittore>Mark Twain</scrittore>
    <titolo>Le avventure di Huckleberry Finn</titolo>
    <anno>1884</anno>
  </libro>
  <libro>
    <autore>Herman Melville</autore>
    <titolo>Moby Dick</titolo>
    <editore/>
    <anno>milleottocentocinquantuno</anno>
  </libro>
</libri>
```

i dati relativi ai singoli libri non sono presenti in modo coerente: per «Moby Dick» l'anno di pubblicazione non è espresso in formato numerico e l'editore non è indicato; per il capolavoro di Mark Twain non è specificato un elemento *editore* e l'autore è definito come contenuto dell'elemento *scrittore*; infine l'anno di pubblicazione di «Terra!» è indicato in forma numerica abbreviata.
Un file di questo tipo risulterebbe di difficile elaborazione da parte di un'applicazione software.

Per un essere umano è possibile comprendere i dati dei tre libri riportati nel documento xml precedente, ma per una applicazione che dovesse ricevere come dato di input tale documento e che dovesse analizzarlo per elaborarne i dati, ci sarebbero delle difficoltà. Ad esempio potrebbe non essere in grado di determinare l'anno di pubblicazione del primo e del terzo libro.

Poiché, come detto, i documenti xml sono utilizzati per lo scambio di dati fra applicazioni, vi è l'esigenza di stabilire delle "regole" che un certo documento xml deve rispettare in modo che i dati in esso contenuti siano interpretabili dalle diverse applicazioni.

Affinché due applicazioni possano scambiarsi dati con documenti xml è necessario che si stabiliscano dei **requisiti** che il documento deve rispettare in termini sia di struttura (quali elementi devono essere presenti) sia di tipi di dati (testo, numero intero ecc..).

Il metodo per verificare che un documento XML rispetti determinati requisiti è chiamato **validazione**. La validazione di un documento viene ottenuta sempre rispetto ad un altro documento che contiene, appunto, "i requisiti da rispettare ("le regole") chiamato **schema XSD**.

La definizione corretta di **XSD (XML Schema Definition)** è che esso è un linguaggio (anche esso definito con XML, nel senso che utilizza appositi tag definiti con XML) che definisce la **grammatica** (le regole sintattiche) di un documento XML.

Con questo linguaggio si realizzano dei documenti (chiamati **schema XSD**, sono file con estensione .xsd) all'interno dei quali si stabiliscono le "regole" che un documento XML deve rispettare.

Una volta stabilite le regole (quindi lo schema XSD), un documento XML è **valido** (rispetto ad uno specifico schemaXSD) se rispetta tali regole.

Cosa si **definisce** in uno schema XSD?

- Quali elementi che deve contenere un documento XML e la loro relazione gerarchica
- Quali eventuali attributi vi devono essere per ogni elemento
- Il numero e l'ordinamento degli elementi
- Gli eventuali elementi vuoti
- Il tipo di dato contenuto negli elementi e negli attributi.
- Eventuali valori predefiniti o costanti degli elementi e degli attributi.

Cosa deve **contenere** un XSD schema?

Deve contenere all'inizio un **elemento radice chiamato schema** con il seguente **attributo predefinito** che riferisce il file contenente gli **elementi** e i **tipi di dato** standard utilizzabili:

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

Inoltre, ogni documento **XML** dovrebbe dichiarare il file contenente lo schema rispetto a cui è stato realizzato. Questo avviene aggiungendo nell'elemento radice i seguenti **attributi predefiniti:**

xmlns:xsi=”http//www.w3.org/XMLSchema-instance”

xsi:noNamespaceSchemaLocation=”…….”

Qui va inserito il pathname del file XSD che contiene lo schema XSD. Può anche essere un URL

Esempio p. 100

ESEMPIO

Un possibile schema XSD per un documento XML che rappresenta un singolo libro potrebbe essere il seguente:

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="libro">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="autore" type="xs:string"/>
        <xs:element name="titolo" type="xs:string"/>
        <xs:element name="editore" type="xs:string"/>
        <xs:element name="anno" type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Un esempio di documento XML valido rispetto allo schema mostrato è il seguente:

```
<?xml version="1.0" ?>
<libro
  xmlns:xsi="http://www.w3.org/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="libro.xsd">
    <autore>Stefano Benni</autore>
    <titolo>Terra!</titolo>
    <editore>Feltrinelli</editore>
    <anno>1983</anno>
</libro>
```

La validazione di un documento XML rispetto ad uno schema può avvenire attraverso opportuni strumenti software (ad esempio Notepad++, oppure attraverso risorse online, ad esempio: http://www.utilities-online.info/xsdvalidation/#.W_LljuhKjIU)

(lo strumento di validazione per Notepad++ fa parte di un plugin aggiuntivo dell'applicazione chiamato XMLtools. In questo video viene mostrato come aggiungere il plugin a Notepad++: https://www.youtube.com/watch?v=hvIKAD1vMCo).

Per validare in Notepad++ un documento XML, si esegue il seguente comando nel menu: plugin/xmlTools/validate Now.

Se nel documento XML non è presente il riferimento al file XSD, Notepad++ chiede di indicare il pathnme del file XSD che contiene lo schema XSD.

Un documento XML **valido** è un documento **ben formato** che rispetta le regole dello schema che riferisce.

Progettare un linguaggio XML significa definirne lo schema, ossia progettare lo schema XSD che stabilisce la sintassi che ogni documento XML deve rispettare per appartenere al linguaggio.

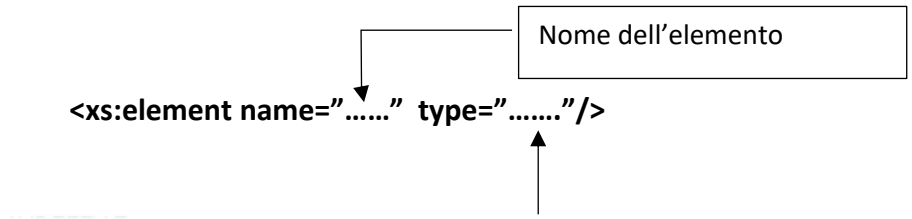
OSSERVAZIONE: esistono altri due strumenti, alternativi all' XSD Schema, per definire la grammatica dei documenti XML. Tali due strumenti si chiamano: **DTD** (Document Type Definition) e **RELAX NG** (REgular Language for XML Next Generation), ma noi ora impareremo ad utilizzare XSD Schema

La definizione degli elementi semplici e degli attributi in un XSD Schema

Cosa è un elemento XML **semplice**?

Un elemento XML semplice è un elemento che può contenere solo testo. Non può contenere altri elementi e non può contenere attributi.

A. Si può **imporre** che il testo contenuto rappresenti: **stringhe, numeri decimali, interi, una data o un tempo**. La sintassi nell' **XSD Schema** è la seguente:



xs:string	Stringhe di caratteri
xs:decimal	Valori numerici non necessariamente interi
xs:integer	Valori numerici interi
xs:date	Data nel formato aaaa-mm-gg (a: anno, m: mese, g: giorno)
xs:time	Ora nel formato oo-mm-ss (o: ora, m: minuti, s: secondi)

XML

```
<?xml version="1.0" ?>

<dataNascita>1975-07-23</dataNascita>
```

XSD Schema

```
<?xml version="1.0" ?>

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

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

<xs:element name="dataNascita" type="xs:date"/>

<xs:element name="eta" type="xs:integer" />

</xs:schema>
```

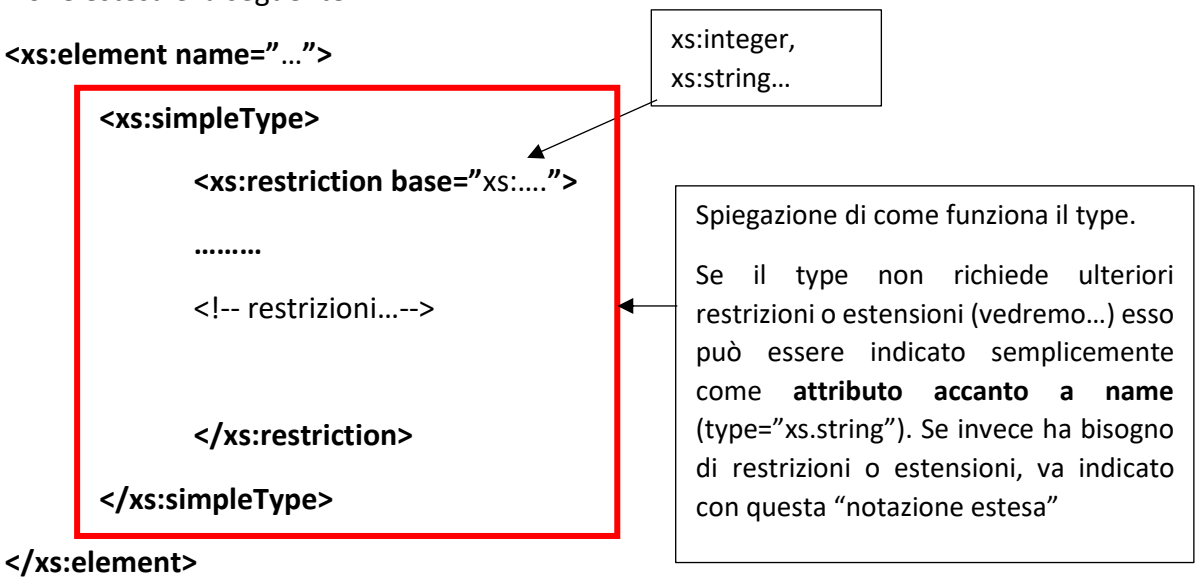
- Nell'XSD schema si possono indicare, per gli elementi semplici:
  1. valori di default *(default="rosso"*): se il documento XML non assegna un valore all'elemento, esso assume il valore di default
  2. valori costanti *(fixed="3.14159"):* il documento XML è valido solo se l'elemento assume il valore.

Non si può definire contemporaneamente per uno stesso elemento valori di default e valori costanti (neppure se sono uguali).

- Lo schema XSD consente di imporre delle **restrizioni** al tipo base di dato indicato (numero di cifre, numero di caratteri ecc..).

  Per impostare le restrizioni ad un tipo di base è necessario utilizzare la notazione stesa per definire un elemento semplice. La notazione che abbiamo utilizzato finora è quella non estesa.

  La notazione estesa è la seguente:



  a. Restrizioni per il tipo di base **numerico (xs:decimal, xs:integer)**. Possono indicare: il valore massimo Incluso o escluso), il valore minimo (incluso o escluso), il numero massimo di cifre, il numero massimo di cifre decimali:

TABELLA 3

xs:fractionDigits	Massimo numero di cifre decimali
xs:maxExclusive xs:maxInclusive	Valore massimo (rispettivamente escluso e incluso)
xs:minExclusive xs:minInclusive	Valore minimo (rispettivamente escluso e incluso)
xs:totalDigits	Numero esatto di cifre

  b. Restrizioni per il tipo di base stringa di caratteri. Possono indicare: il numero massimo, minimo, o esatto di caratteri:0

TABELLA 4

xs:length	Numero esatto di caratteri
xs:maxLength	Numero massimo di caratteri
xs:minLength	Numero minimo di caratteri

xs:pattern	espressione regolare

Esempio per la restrizione pattern per il codice fiscale

```
<xs:element name="CF">
<xs:simpleType>
        <xs:restriction base="xs:string">
                <xs:pattern value="[A-Z0-9]{6}[0-9]{2}[A-Z]{1}[0-9]{2}[A-Z]{1}[0-9]{3}[A-Z]{1}"/>
        </xs:restriction>
</xs:simpleType>
</xs:element>
```

6 CARATTERI ALFANUMERICI

2 CARATTERI NUMERICI

....

CARATTERI VALIDI

QUANTITA' DI CARATTERI

elemento semplice valido:

<CF>LNNMRC75L22B149A<CF>

Osservazione: il file XSD non controlla che il l'ultima lettera, che è un codice di controllo, sia corretta. Controlla solo che sia una lettera maiuscola.

c. Restrizioni **enumerative** per il tipo di base **stringa di caratteri**. Consente di indicare un elenco di stringhe valide.

```
<xs:restriction base="……">
        <xs:enumeration value="…valore1…"/>
        <xs:enumeration value="…valore2…"/>
        ……
</xs:restriction>
```

Esempio di schema che applica alcune restrizioni per alcuni elementi semplici:

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

  <xs:element name="voto">
        <xs:simpleType>
            <xs:restriction base="xs:integer">
               <!-- restrizioni-->
                    <xs:minExclusive value="1"/>
                    <xs:maxExclusive value="10"/>
            </xs:restriction>
        </xs:simpleType>
  </xs:element>


 <xs:element name="CAP">
        <xs:simpleType>
            <xs:restriction base="xs:string">
               <!-- restrizioni-->
                    <xs:length value="5"/>
            </xs:restriction>
        </xs:simpleType>
  </xs:element>



 <xs:element name="indirizzo">
        <xs:simpleType>
            <xs:restriction base="xs:string">
               <!-- restrizioni-->
                    <xs:minLength value="5"/>
                    <xs:maxLength value="100"/>
            </xs:restriction>
        </xs:simpleType>
  </xs:element>

<xs:element name="colore">
         <xs:simpleType>
              <xs:restriction base="xs:string">
                  <xs:enumeration value="bianco"/>
                   <xs:enumeration value="blu"/>
                   <xs:enumeration value="verde"/>
              </xs:restriction>
         </xs:simpleType>
</xs:element>


</xs:schema>
```

Un elenco enumerativo (così come qualsiasi altro elemento) può essere utilizzato per definire un tipo (type) ed utilizzato quindi, all'interno dello schema XSD per **definire il tipo** di un altro elemento.

Esempio

le restrizioni applicate precedentemente nella definizione dell'elemento colore, le utilizzo per definire un symple type colore:

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="Colore">
      <xs:restriction base="xs:string">
        <xs:enumeration value="bianco"/>
         <xs:enumeration value="giallo"/>
        <xs:enumeration value="rosso"/>
        <xs:enumeration value="verde"/>
        <xs:enumeration value="blu"/>
      </xs:restriction>
    </xs:simpleType>

  <xs:element name="coloreCarrozzeria" type="Colore"/>
  <xs:element name="coloreInterni" type="Colore "/>
  <xs:element name="coloreSpecchietto" type="Colore "/>
</xs: schema>
```

Attenzione: per creare un nuovo "tipo di dato" assegno il nome al simpleType, il tag elemento non c'è.

Nello schema ora posso definire elementi di tipo nomeColore (da usare poi per costruire elementi complessi)

Definizione degli attributi

Si ricorda che gli elementi semplici non possono avere attributi. In uno schema XSD, si possono definire degli **attributi** per un elemento, in tal caso l'elemento sarà un elemento **complesso**, non un elemento semplice.

La definizione di un attributo è analoga a quella degli elementi semplici:

<xs:attribute name="….." type="…."/>

Un attributo è di default facoltativo, esso può essere reso obbligatorio valorizzando **l'attributo predefinito "use"** come **rquired** (l'attributo use è un attributo predefinito usato nella definizione degli attributi)

<xs:attribute name="….." type="…." use="required**"/>**

LA DEFINIZIONE DEGLI ELEMENTI COMPLESSI

Cosa è un elemento complesso XML?

Un elemento complesso è un elemento che ha una delle seguenti caratteristiche:

1. è un elemento che contiene altri elementi
2. è un elemento vuoto. Eventualmente con attributi
3. è un elemento che contiene solo testo (quindi che sarebbe semplice) ma con attributi
4. è un elemento che contiene sia testo che altri elementi

Un elemento complesso può essere definito, all'interno di uno schema XSD in maniera diretta, ma generalmente viene definito mediante un **tipo (type)** che permette di definire più elementi complessi.

ESEMPIO 1 A: definizione di elementi complessi utilizzando la definizione di un type

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:complexType name="tipoPersona">
            <xs:sequence>
                <xs:element name="cognome" type="xs:string"/>
                <xs:element name="nome" type="xs:string"/>
                <xs:element name="dataNascita" type="xs:date"/>
                <xs:element name="luogoNascita" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    <xs:element name="studente" type="tipoPersona"/>
    <xs:element name="docente" type="tipoPersona"/>
</xs:schema>
```

L'indicatore <xs:sequence> impone che gli elementi complessi *studente* e *docente* contengano ciascuno gli elementi semplici *cognome, nome, dataNascita, luogoNascita* nell'ordine specificato dal documento XSD.

ESEMPIO 1 B: definizione di elemento complesso con restrizioni

Definizione di un elemento complesso (indirizzo) contenente elementi semplici in un ordine preciso. Gli elementi contenuti sono: via, numero (deve essere>1), cap (deve essere di 5 caratteri), citta.

XSD schema	XML
<?xml version ="1.0" ?> **<xs:schema** **xmlns:xs=**"http://www.w3.org/2001/XMLSchema**">** **<xs:element name=**"indirizzo"**>** **<xs:complexType>** **<xs:sequence>** **<xs:element name=**"via" **type=**"xs:string"**/>** **<xs:element name=**"numero"**>** **<xs:simpleType>** **<xs:restriction base=**"xs:integer"**>** **<xs:minInclusive value=**"1"**/>** **</xs:restriction>** **</xs:simpleType>** **</xs:element>** **<xs:element name=**"cap"**>** **<xs:simpleType>** **<xs:restriction base=**"xs:string"**>** **<xs:length value=**"5"**/>** **</xs:restriction>** **</xs:simpleType>** **</xs:element>** **<xs:element name=**"citta" **type=**"xs:string"**/>** **</xs:sequence>** **</xs:complexType>** **</xs:element>** **</xs:schema>**	Valido: <?xml version="1.0" ?> <indirizzo> <via>Via dei tigli</via> <numero>8</numero> <cap>25055</cap> <citta>Lovere</citta> </indirizzo> non valido <?xml version="1.0" ?> <indirizzo> <via>Via dei tigli</via> <numero>8</numero> <citta>Lovere</citta> <cap>25055</cap> </indirizzo> invertiti non valido <?xml version="1.0" ?> <indirizzo> <via>Via dei tigli</via> <numero>8</numero> <cap>2505</cap> <citta>Lovere</citta> </indirizzo> 4 cifre

ESEMPIO 2: elemento vuoto

(senza attributi)

Il seguente schema XML:

```
<?xml version ="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="elementoVuoto">
                <xs:complexType>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Consente di validare:

```
<elementoVuoto/>
```

e anche

```
<elementoVuoto></elementoVuoto>
```

Attenzione, se qui metto un carattere (ad esempio return), l'elemento non viene validato. Invece se si definisce un elemento semplice con nessun testo, esso viene validato sia con testo che senza testo.

(con attributi)

```
<?xml version ="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="elementoVuoto">
                <xs:complexType>
                        <xs:attribute name="attributo" type="xs:string"/>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

consente di validare

```
<elementoVuoto attributo="attributo di prova"></elementoVuoto>
```

ESEMPIO 3: elemento che contiene **solo testo** (ad esempio xs:decimal) ma con attributi.

NOTA BENE: quando dico "contiene solo testo" intendo dire "che non contiene altri elementi", ossia il testo può anche essere di tipo numerico (xs:integer, xs:decimal….), l'elemento in questione sarebbe semplice se non fossero definiti per esso degli attributi.

Questo è dunque il caso di un elemento chiamato complesso (perché ci sono gli attributi) con contenuto semplice (solo testo di tipo decimal). Nell'esempio, l'elemento da realizzare lo chiamiamo misura.

Per rendere più chiara la situazione partiamo con un elemento misura che sia semplice. Poi vediamo come dobbiamo fare per aggiungere un attributo, azione che rende l'elemento misura complesso.

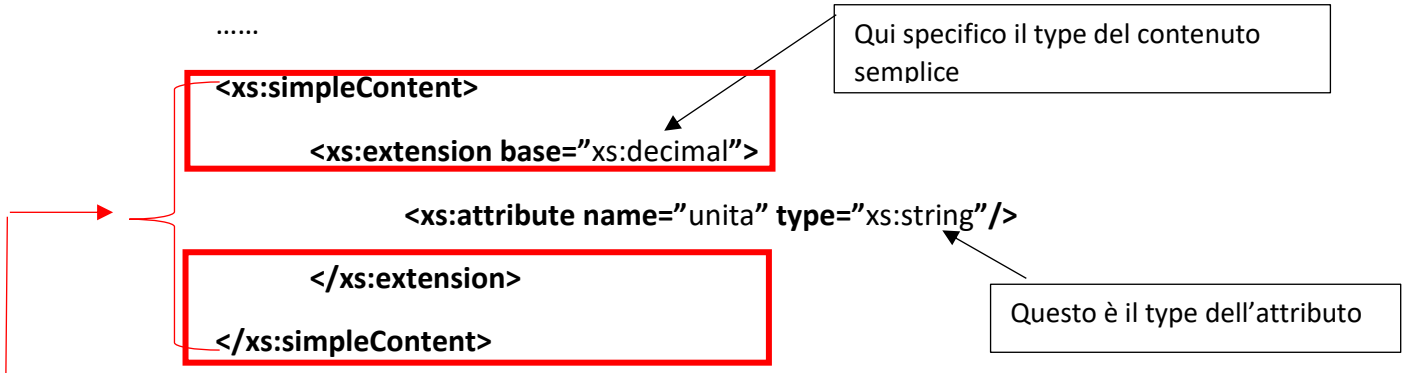
STEP1: definizione di un elemento semplice misura. Contiene solo testo di tipo decimal

XSD schema	XML
<?xml version="1.0" ?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> <xs:element name="misura" type="xs:decimal"/> </xs:schema>	Valido <misura>0.01</misura> Valido <misura>100</misura> Non valido <misura>100 cm</misura>

STEP 2: supponiamo di voler aggiungere l'attributo *unita* all'elemento *misura*. L'attributo *unita* è di tipo stringa e serve per indicare l'unità di misura dell'elemento.

Questa modifica rende l'elemento *misura* complesso poiché viene aggiunto un attributo.

Poiché l'elemento misura non contiene altri elementi, il suo contenuto è comunque semplice (solo testo di tipo xs:decimal). L'attributo e il contenuto semplice vanno indicati con la seguente struttura:



Infine si decide di rendere obbligatorio l'attributo *unita* specificando nella sua definizione l'attributo *use="required"*.

xs:extension base="xs:type" estende un tipo (type) o un elemento (sia semplice che complesso) aggiungendo uno o più attributi o elementi. Il concetto di estensione è analogo a quello dell'ereditarietà

Esempi di estensione: https://www.w3schools.com/xml/el_extension.asp

Quindi lo schema XSD finale è il seguente

XSD schema	XML
<?xml version="1.0" **?>** **<xs:schema xmlns:xs="**http://www.w3.org/2001/XMLSchema"> **<xs:element name="**misura**">** **<xs:complexType>** **<xs:simpleContent>** **<xs:extension base="**xs:decimal**">** **<xs:attribute name="**unita" **type="**xs:string" **use="**required**"/>** **</xs:extension>** **</xs:simpleContent>** **</xs:complexType>** **</xs:element>** **</xs:schema>**	Valido **<misura unita="**cm**">**0.01**</misura>** Valido **<misura unita="**cm**">**100**</misura>** Non valido **<misura>**100**</misura>** Manca l'attributo unita

Anche agli attributi, come agli elementi semplici, è possibile applicare le restrizioni aggiungendo

<xs:attribute name="..">

<xs:simpleType> //perché l'attributo, di per se , è un elemento semplice

<xs:restriction base="…">

<!-- restrizioni…. -->

</xs:restriction>

<xs:simpleType>

<xs:attribute name="..">

COME AGGIUNGERE UN ATTRIBUTO A UN ELEMENTO COMPLESSO (NON C'E' SPIEGATO BENE SUL LIBRO)

Si premette che un attributo si definisce sempre come elemento semplice

<xs:attribute name="attributoX" type="xs:string">

anche ad esso, come per gli elementi semplici, è possibile aggiungere delle restrizioni.

Si ricorda che l'aggiunta di un attributo rende un elemento complesso.

Per aggiungere un attributo ad un elemento (sia ad un elemento di partenza semplice, sia già complesso) si opera un' **"estensione".**

La strategia è la seguente. Supponiamo di voler aggiungere all'elemento complesso libro (che contiene gli elementi autore, titolo, prezzo) l'attributo "genere":

1. Nello schema XSD si definisce un tipo di elemento complesso "tipoLibro" contenente i vari elementi (autore, titolo, prezzo)

2. Si definisce, nello stesso schema XSD, l'elemento complesso "libro" di tipo "tipoLibro" estendendolo con l'aggiunta dell'attributo "genere"

schema XSD

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

    <!--definisco tipo di dato complesso "tipoLibro" -->
    <xs:complexType name="tipoLibro">
        <xs:sequence>
            <xs:element name="autore" type="xs:string"/>
            <xs:element name="titolo" type="xs:string"/>
            <xs:element name="prezzo" type="xs:float"/>
        </xs:sequence>
    </xs:complexType>

    <!--definisco l'elemento "libro" di tipo "tipoLibro"
    estendendo tale tipo con l'attributo "genere"-->
    <xs:element name="libro">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="tipoLibro">
                    <xs:attribute name="genere" type="xs:string" use="required"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

XML

```
<?xml version="1.0" ?>
<libro genere="romanzo">
    <autore>Alessandro Manzoni</autore>
    <titolo>I promessi sposi</titolo>
    <prezzo>10</prezzo>
</libro>
```

ESEMPIO 4: elemento *messaggio* che contiene sia testo che altri elementi (un codice, data, ora, un luogo)

Per definire un elemento XML che possa contenere sia testo (di qualsiasi tipo) che altri elementi, è necessario impostare, nello schema XSD, l'attributo *mixed="true"* nella definizione dell'elemento complesso.

XSD schema	XML
<?xml version="1.0" ?> **<xs:schema** **xmlns:xs="**http://www.w3.org/2001/XMLSchema**">** **<xs:element name="**messaggio**">** **<xs:complexType mixed="**true**">** **<xs:sequence>** **<xs:element name="**codice**" type="**xs:string**/>** **<xs:element name="**data**" type="**xs:date**/>** **<xs:element name="**ora**" type="**xs:time**/>** **<xs:element name="**luogo**" type="**xs:string**/>** **</xs:sequence>** **</xs:complexType>** **</xs:element>** **</xs:schema>**	Valido <?xml version="1.0" ?> <**messaggio**> La spedizione <**codice**>132333</**codice**> è stata inviata in data <**data**>2018-10-10</**data**> alle ore <**ora**>15:30:00</**ora**> al <**luogo**>Zurigo</**luogo**> </**messaggio**>

Elementi complessi contenenti altri elementi in cui l'ordine degli elementi contenuti non è rilevante.

Oltre a <**sequence**> esiste un altro indicatore che consente di indicare una sequenza di elementi, tale indicatore è **<all>.** Quest'ultimo indicatore richiede che in un documento XML tutti gli elementi presenti all'interno dell'elemento complesso siano presenti ma l'ordine in cui sono presenti è irrilevante.

Esempio: rifacendo lo schema XSD indirizzo.xsd sostituendo <sequence> con <all> verificare il suo funzionamento

Elementi complessi contenenti altri elementi fra i quali ne va scelto uno (solo) fra più elementi possibili

Oltre a <**sequence**> e ad **<all>** esiste un altro indicatore che consente di indicare una sequenza di elementi, tale indicatore è **<choice>**. Quest'ultimo indicatore consente di indicare una sequenza di più elementi ma richiede che in un documento XML sia presente uno solo fra gli elementi dell'elenco.

XSD schema	XML
<?xml version="1.0" ?> **<xs:schema** **xmlns:xs="http://www.w3.org/2001/XMLSchema">** **<xs:element name="**riferimentoCliente**">** **<xs:complexType>** **<xs:choice>** **<xs:element name="**codiceFiscale**">** **<xs:simpleType>** **<xs:restriction base="**xs:string**">** **<xs:length value="**16**"/>** **</xs:restriction>** **</xs:simpleType>** **</xs:element>** **<xs:element name="**partitaIVA**">** **<xs:simpleType>** **<xs:restriction base="**xs:string**">** **<xs:length value="**11**"/>** **</xs:restriction>** **</xs:simpleType>** **</xs:element>** **</xs:choice>** **</xs:complexType>** **</xs:element>** **</xs:schema>**	Valido: **<riferimentoCliente>** **<partitaIVA>**12345432123**</partitaIVA>** **</riferimentoCliente>** Valido: **<riferimentoCliente>** **<codiceFiscale>**LNRGBF56L32B231Q**</codiceFiscale>** **</riferimentoCliente>** Non valido**:** **<riferimentoCliente>** **<codiceFiscale>**LNRGBF56L32B231Q**</codiceFiscale>** **<partitaIVA>**12345432123**</partitaIVA>** **</riferimentoCliente>**

E' possibile, nell' XSD, inserire elemnti <sequence>, <all> e <choice> uno dentro l'altro gerarchicamente.

Per esempio un type tipoIndirizzo che consente, a scelta di indicare la via o la piazza, è il seguente:

```
<xs:complexType name="tipoIndirizzo">
        <xs:sequence>
            <xs:element name="comune" type="xs:string"/>
            <xs:choice>
                <xs:element name="via" type="xs:string"/>
                <xs:element name="piazza" type="xs:string"/>
            </xs:choice>
            <xs:element name="numeroCivico" type="xs:string"/>
            <xs:element name="CAP" type="tipoCAP"/>
        </xs:sequence>
    </xs:complexType>
```

con tipoCAP che è un tipo xs:string semplice con la seguente restrizione:

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

Indicare più occorrenze di un elemento

Quando si vuole consentire che all'interno di un elemento complesso che contiene più elementi, uno o più degli elementi contenuti possa avere **più occorrenze**, si utilizzano gli indicatori *minOccurs* e *maxOccurs* nella definizione dell'elemento:

Per ottenere un numero di occorrenze illimitato si assegna il valore *unbounded* all'attributo *maxOccurs.* Indicando minOccurs="0", nell'XML l'elemento contenuto può essere omesso.

XSD schema	XML
<?xml version="1.0" **?>** **<xs:schema xmlns:xs=**"http://www.w3.org/2001/XMLSchema"**>** **<xs:element name=**"persona"**>** **<xs:complexType>** **<xs:sequence>** **<xs:element name=**"cognome" **type=**"xs:string"**/>** **<xs:element name=**"nome" **type=**"xs:string"**/>** **<xs:element name=**"figlio" **type=**"xs:string" **minOccurs=**"0" **maxOccurs=**"unbounded"**/>** **</xs:sequence>** **</xs:complexType>** **</xs:element>** **</xs:schema>**	XML Valido: **<?xml version=**"1.0" **?>** **<persona>** **<cognome>**Lilla**</cognome>** **<nome>**Pennario**</nome>** **<figlio>**Rino**</figlio>** **<figlio>**Peppe**</figlio>** **</persona>** XML Valido: **<?xml version=**"1.0" **?>** **<persona>** **<cognome>**Lilla**</cognome>** **<nome>**Pennario**</nome>** **</persona>** Valido**:** **<?xml version=**"1.0" **?>** **<persona>** **<cognome>**Lilla**</cognome>** **<nome>**Pennario**</nome>** **<figlio>**Rino**</figlio>** **</persona>**

Si potrebbe (e sarebbe meglio) definire un tipoPersona in cui l'elemento figlio è a sua volta di tipo tipoPersona. (fare per esercizio).

Soluzione:

XSD schema

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:complexType name="tipoPersona">
                <xs:sequence>
                    <xs:element name="cognome"
type="xs:string"/>
                        <xs:element name="nome"
type="xs:string"/>
                        <xs:element name="figlio"
type="tipoPersona" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="persona" type="tipoPersona"/>
</xs:schema>
```

XML

XML Valido:

```
<?xml version="1.0" ?>
<persona>
  <cognome>Lilla</cognome>
   <nome>Pennario</nome>
   <figlio>
        <cognome>Pinna</cognome>
        <nome>Nando</nome>
   </figlio>
</persona>
```

XML Valido:

```
<?xml version="1.0" ?>
<persona>
        <cognome>Pinna</cognome>
        <nome>Luigi</nome>
</persona>
```

Valido:

```
<?xml version="1.0" ?>
<persona>
  <cognome>Lilla</cognome>
   <nome>Pennario</nome>
   <figlio>
        <cognome>Pinna</cognome>
        <nome>Nando</nome>
   </figlio>
   <figlio>
        <cognome>Pinna</cognome>
        <nome>Luisa</nome>
   </figlio>
</persona>
```

COME ESTENDERE UN TIPO DI DATO

A partire da un tipo di dato (semplice o complesso) è sempre possibile ottenere un altro tipo di dato con l'operazione di "estensione", come abbiamo visto nel caso dell'aggiunta di attributi.. L' estensione consente di creare un nuovo tipo di dato che aggiunge attributi o ulteriori elementi al tipo di dato complesso di partenza.

Esempio: partiamo da un tipo di dato complesso "tipoPersona" che contiene solo gli elementi semplici cognome e nome:

XSD schema

```
<xs:complexType name="tipoPersona">
    <xs:sequence>
        <xs:element name="cognome" type="xs:string"/>
        <xs:element name="nome" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

XML

```
<?xml version="1.0"?>
<persona>
    <cognome>Pinna</cognome>
    <nome>Luciano</nome>
</persona>
```

Se vogliamo estendere il tipo di dato "tipoPersona" aggiungendo un ulteriore elemento, ad esempio una sequenza di elementi "cane" di tipo stringa, che indica gli eventuali cani di quella persona, la sintassi è la seguente:

XSD schema

```
<xs:complexType name="tipoPersona">
    <xs:sequence>
        <xs:element name="cognome" type="xs:string"/>
        <xs:element name="nome" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="tipoPersonaConCan
    <xs:complexContent>
        <xs:extension base="tipoPersona">
            <xs:sequence>
                <xs:element name="cane" type="xs:string"
                minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="persona" type="tipoPersonaConCani"/>
```

indica che il contenuto è complesso

tipo di dato da estendere

sequenza dei nuovi elementi da aggiungere

XML valido

```
<persona>
    <cognome>Pinna</cognome>
    <nome>Luciano</nome>
    <cane>Fido</cane>
    <cane>Lido</cane>
</persona>
```

I TIPI DI DATO PREDEFINITI

Vediamoli una volta tutti così poi sappiamo che ci sono e quando ci servono li usiamo.

Se l'elemento del documento XML conterrà tab o ritorni a capo, il software che analizza e interpreta il documento (questo sw è chiamato **parser** ed è presente in tutti i browser) **sostituirà tali caratteri con degli spazi**.

xs:string	Stringhe di caratteri
xs:normalizedString	Stringhe di caratteri non suddivise su più linee e prive di tabulazioni
xs:token	Stringhe di caratteri non suddivise su più linee, prive di tabulazioni, di spazi iniziali e finali e di spazi ripetuti
xs:float	Valori numerici *floating-point* a 32 bit
xs:double	Valori numerici *floating-point* a 64 bit
xs:decimal	Valori numerici non necessariamente interi
xs:byte xs:unsignedByte	Valori numerici rappresentabili con 8 bit, rispettivamente con segno e senza segno

Da -128 a 127

Da 0 a 255

Il **parser eliminerà** tali caratteri

A cosa servono Integer, non NegativeInteger ecc? Sono di lunghezza arbitraria illimitata. Usali quando non sai quale è il max valore che rappresenterai, altrimenti, se lo sai, usa int, short o long consente prestazioni di calcolo migliori da parte di processori. Quando si svolge il parsing in Java, Integer viene convertito in un tipo di dato BigInteger che è un oggetto la cui dimensione cresce dinamicamente e può quindi contenere numeri interi molto grandi.

xs:short xs:unsignedShort	Valori numerici rappresentabili con 16 bit, rispettivamente con segno e senza segno
xs:int xs:unsignedInt	Valori numerici rappresentabili con 32 bit, rispettivamente con segno e senza segno
xs:long xs:unisgnedLong	Valori numerici rappresentabili con 64 bit, rispettivamente con segno e senza segno
xs:integer	Valori numerici interi
xs:nonNegativeInteger	Valori numerici interi non negativi (compreso lo 0)
xs:negativeInteger	Valori numerici interi negativi
xs:nonPositiveInteger	Valori numerici interi non positivi (compreso lo 0)
xs:positiveInteger	Valori numerici interi positivi
xs:date	Data nel formato aaaa-mm-gg (a: anno, m: mese, g: giorno); un carattere «Z» finale opzionale indica che si tratta della data UTC.
xs:time	Ora nel formato oo:mm:ss (o: ora, m: minuti, s: secondi): i secondi possono assumere un valore non intero; un carattere «Z» finale opzionale indica che si tratta dell'ora UTC.
xs:dateTime	Data/ora nel formato aaaa-mm-ggToo:mm:ss (a: anno, m: mese, g: giorno, o: ora, m: minuti, s: secondi): i secondi possono assumere un valore non intero; un carattere «Z» finale opzionale indica che si tratta dell'ora UTC.
xs:duration	Durata temporale nel formato PaYmMgDToHmMsS (a: anni, m: mesi, g: giorni, o:ore, m: minuti, s:secondi; i secondi possono assumere un valore non intero); le varie sezioni sono opzionali, il carattere «P» inizia obbligatoriamente la rappresentazione del periodo, il carattere «T» inizia obbligatoriamente la parte oraria del periodo.
xs:boolean	Valori *true/false*
xs:hexBinary	Sequenze di byte codificate in esadecimale
~~xs:base64Binary~~	~~Sequenze di byte codificate in base-64[4]~~
xs:anyURI	Stringa che rappresenta un URI (*Uniform Resource Identifier*)

Tutti i componenti sono necessari

Tutti i componenti sono necessari

NON tutti i componenti sono necessari

NON L'ABBIAMO VISTO QUINDI TRASCURIAMO PER ORA

URI

URL (uniform resource locator): indica anche come e dove localizzare la risorsa (ad esempio una pagina web)

URN (uniform resource name): indica una risorsa ma non come e dove localizzare la risorsa (ad esempio un ISBN di un libro)

VALIDO: 34AE NON VALIDO : 34A perché il numero binario dovrà essere codificato in una sequenza di byte, quindi deve avere un numero pari di cifre

<data>2018-02-02Z**</data>** indica che è la data UTC (Universal Time Coordinated), quella del meridiano di Greenwich.

<data>2018-02-02-06:00**</data>** specifica un fuso orario di 6 ore in meno rispetto all'ora di Greenwich

L'orario **UTC** (Universal Time Coordinated) indica l'orario di Greenwich. Tale orario in passato veniva indicato anche GTM (Greenwich Meridian Time) oppure indicato con una Z (o anche con Zulu, che nel gergo dei piloti militari indica, appunto la z). Durante l'ora solare (inverno) in Italia l'ora è UTC+1, durante l'ora legale (estate) è UTC+2

ESEMPIO VIAGGIO P. 111

ESEMPIO

Data la seguente definizione dell'elemento complesso *viaggio*:

```
<xs:element name="viaggio">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="mezzo">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="auto"/>
            <xs:enumeration value="treno"/>
            <xs:enumeration value="aereo"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="partenza" type="xs:dateTime"/>
      <xs:element name="luogoPartenza" type="xs:string"/>
      <xs:element name="arrivo" type="xs:dateTime"/>
      <xs:element name="luogoArrivo" type="xs:string"/>
      <xs:element name="durata" type="xs:duration"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

è valido il seguente elemento XML:

```
<viaggio>
  <mezzo>treno</mezzo>
  <partenza>2012-10-21T10:35:10</partenza>
  <luogoPartenza>Livorno</luogoPartenza>
  <arrivo>2012-10-21T14:40:30</arrivo>
  <luogoArrivo>Milano</luogoArrivo>
  <durata>PT4H5M20S</durata>
</viaggio>
```

ESEMPIO RIFERIMENTI P. 112

ESEMPIO

Data la seguente definizione dell'elemento complesso *riferimenti*:

```
<xs:element name="riferimenti">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="riferimento" type="xs:anyURI"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

risultano validi i seguenti elementi XML:

```
<riferimenti>
  <riferimento>http://www.w3schools.com</riferimento>
  <riferimento>http://www.w3.org/xml/schema_intro.asp</riferimento>
</riferimenti>
```

```
<riferimenti>
</riferimenti>
```

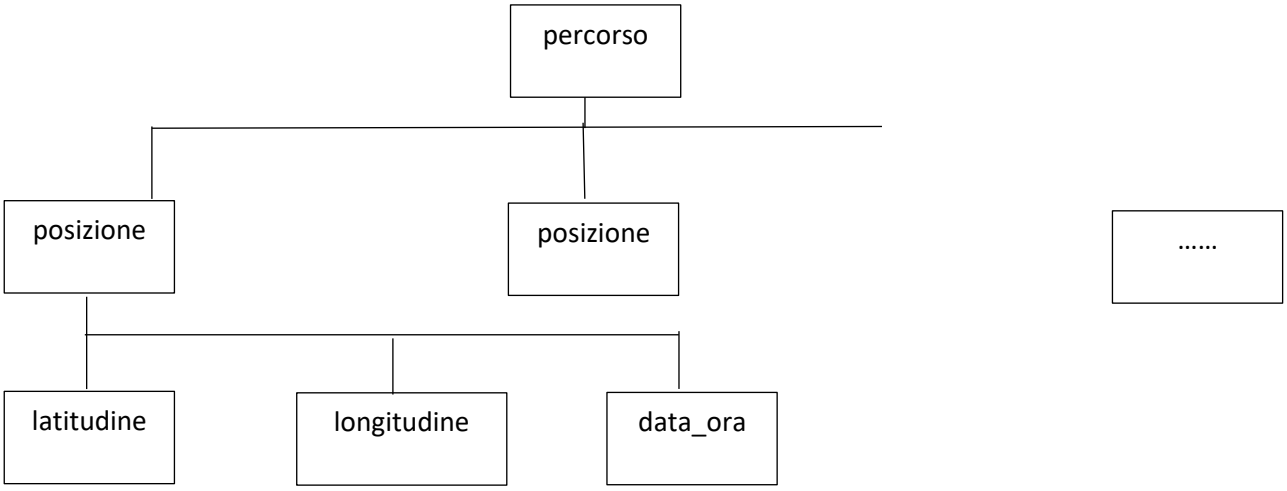
ESEMPIO COMPLETO P.112

PERCORSO

Il percorso di un veicolo sulla superficie terrestre è una lista ordinata di **posizioni geografiche.**

Ogni posizione è formata da una latitudine, longitudine, data e ora.

Ciascuna Latitudine e ciascuna longitudine è a sua volta formata da gradi, minuti, secondi e dall'orientamento (nord/sud per latitudine, est/ovest per longitudine).



Soluzione per xml schema

Definiamo i tipi di dato complessi **tipo_latitudine** e **tipo_longitudine** formati dai dati semplici **gradi, minuti, secondi, orientamento** (ciascuno con le restrizioni).

> Restrizioni tipo_latitudine: gradi (0-90), minuti (0-60), secondi (0-60), orientamento (Nord,sud).
>
> Restrizioni tipo_longitudine: gradi (0-180), minuti (0-60), secondi (0-60), orientamento (Est,Ovest).

Definiamo il tipo di dato **posizione** formato da elementi: latitudine (tipo latitudine), longitudine (tipo longitudine), dataOra (semplice di tipo xs:dateTime)

Definiamo l'elemento **percorso** come elenco infinito di elementi posizione.

Fare esercizi del libro a p. 186,187 135,136: (in giallo le pagine si riferiscono al libro vecchio)

28 21 giornale: (tranne l'elemento "Intestazione" che è ancora un titolo e quindi non ha senso)

30 23 Verifica quiz: (punteggio minimo=0, numero di risposte minimo=3, massimo=5)

32, 30 Cassonetti: tranne la parte in Java. Latitudine in decimale è un numero compreso fra 90 e -90 con esattamente 6 cifre decimali. Longitudine è un numero compreso fra -180 e 180 con esattamente 6 cifre decimali. Esprimere l'unità di misura del volume (mc) come attributo che può assumere solo il valore mc.

33 31:autonoleggio (tranne la parte in Java)

33: percorso (tranne la parte in Java)