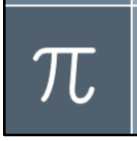
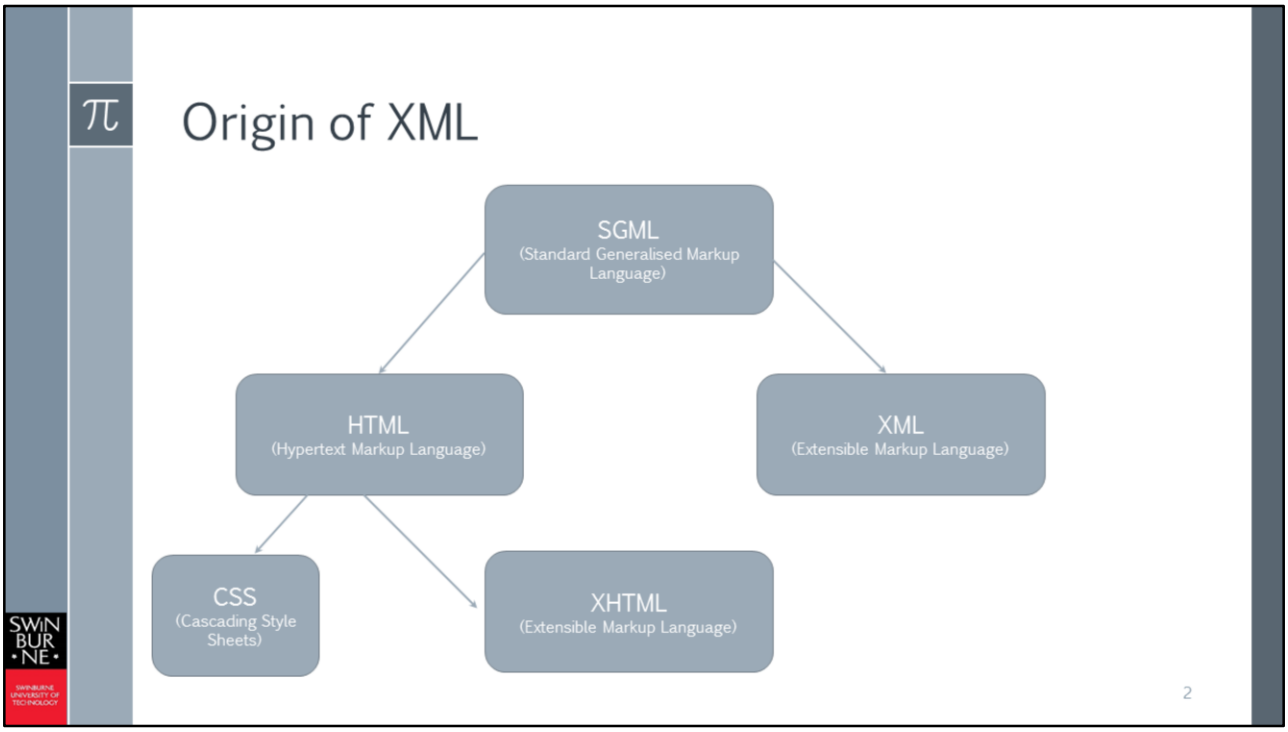
Semi-Structured Data

Tree-Structures: XML and JSON



This module presents an introduction to the most common formats of semi-structured data, XML and JSON. XML is still used in many web-based technologies, and JSON has become a popular data format for NoSQL databases and web-based data interchange.



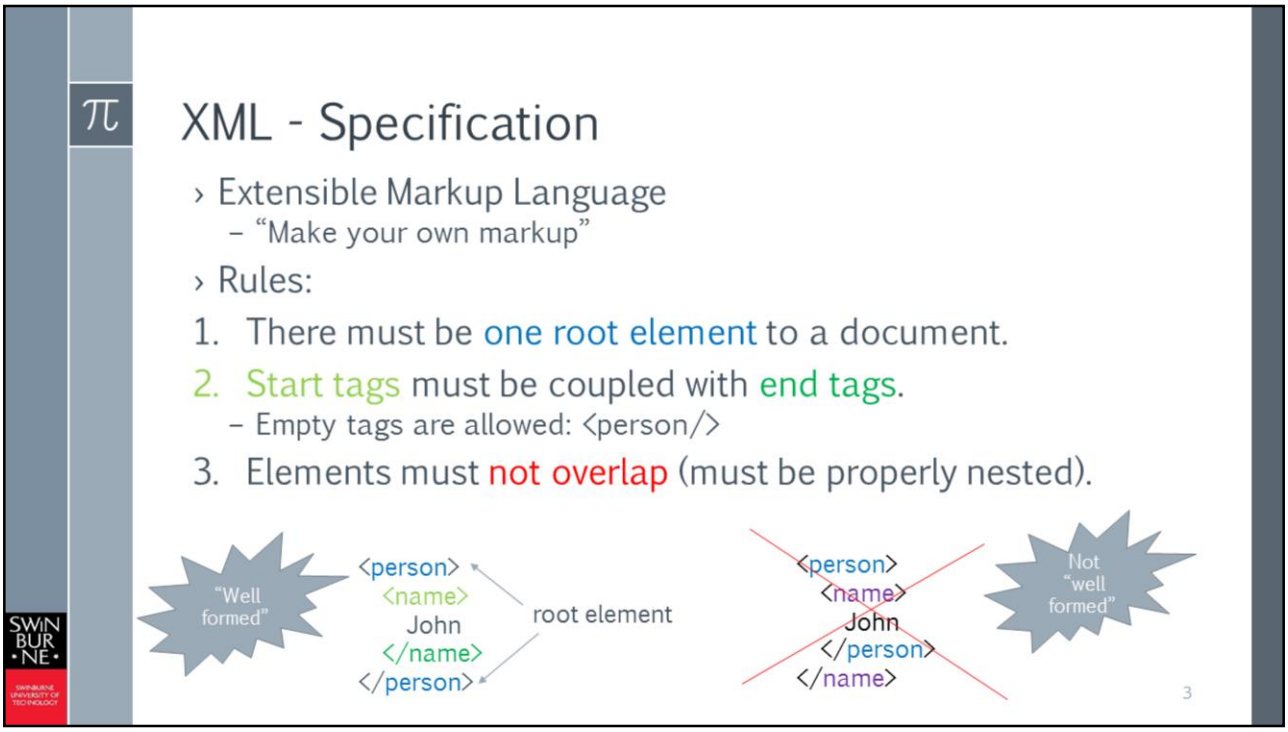
SGML was conceived in the mid-1980s as part of the Electronic Data Interchange concept. Essentially, companies began to realise that sending each other letters in the post meant a lot of data entry work, something that could easily be remedied if we had a machine-readable data format. This would allow software programs to exchange documents over a network. This happened long before the time of e-mail, which was not used among the general public before 1995. However, SGML was not intended to be used in an e-mail-like system. Its purpose was to facilitate the exchange of legal documents such as orders, invoices, receipts and tax returns between companies and between companies and government organisations. The documents needed to be humanly legible but were primarily intended for electronic readers. This is why 'markup' was used – pieces of information encased in tags that explained their meaning to humans and machines alike.

SGML is still used in companies for this purpose, but it is very complex, embedding the document structure description with the content. When Tim Berners Lee developed the World Wide Web in 1990, he needed a simpler and more specialised markup for documents that browsers could show to people and HTML was proposed. The dawn of the web started a competition between browser manufacturers. They scrambled to add more features to the browsers so that web sites would look more interesting. This started the

first browser wars between Netscape Navigator and Microsoft Internet Explorer. In these times, HTML tags were added by both companies on the fly and documents would display differently on the two browsers. Many formatting tags were invented, which is against the principle of separating the data from its representation. Also, HTML 1.0 was not entirely XML-compliant; start tags with no matching end tags were allowed.

XHTML was created to remedy this, and the corresponding formatting instructions were shifted into CSS files. For backward compatibility, browsers can still show non-compliant HTML.

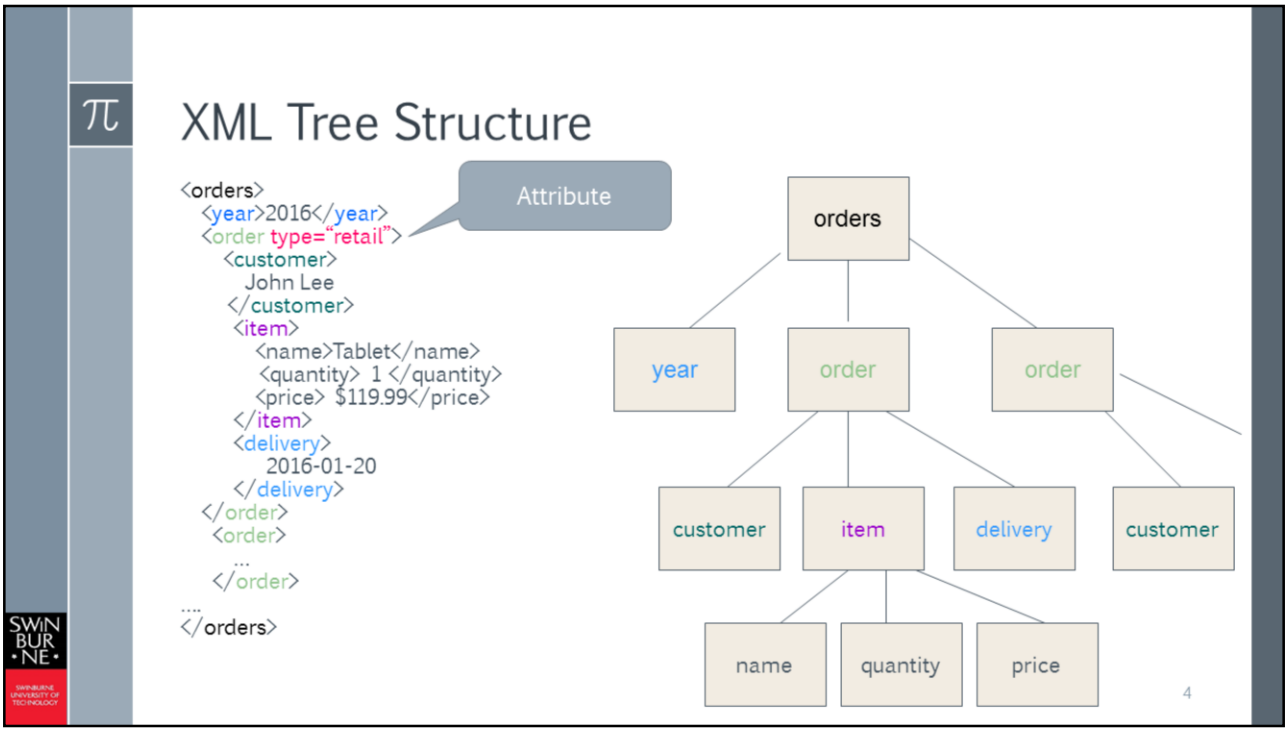
Since 1996, XML was developed as a lightweight equivalent of SGML which could be used for electronic data exchanges on the emerging World Wide Web. Many web technologies rely on XML, for example the SOAP protocol which is a core part of Web Services. There are some widely used discipline-specific XML implementations such as MathML and CML (Chemistry ML).



XML is a markup language. Like html, it 'marks' every piece of data by placing it between a start and an end tag. The result, the value and its tags, is called an element. We can have elements with simple string values in them or elements that contain further elements.

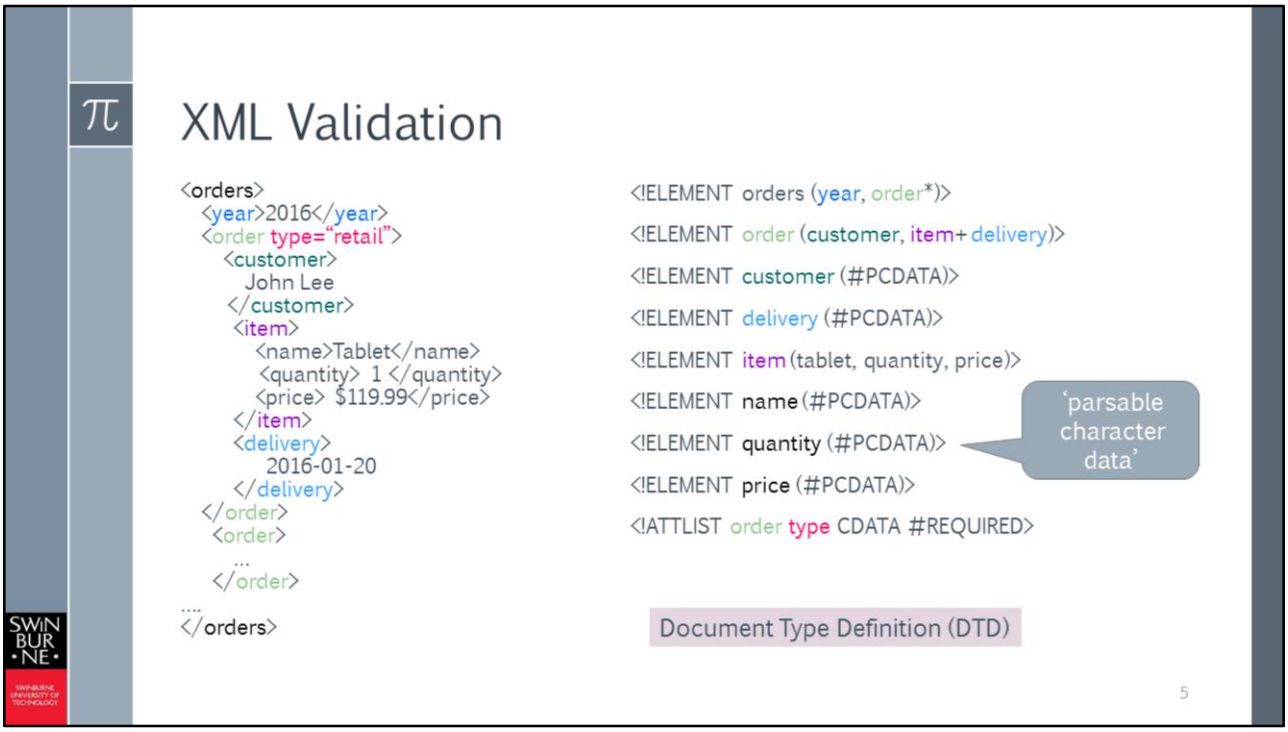
Unlike HTML, where we have predefined tag names such as head and body, in XML we are allowed to define our own tags. This is why XML is extensible. We are free to create our own tags and elements with any structure we like. We only have to observe three rules. First, the document has to have a single root element that encases all others. Second, each start tag must have an end tag and third, the start and end tags of different elements must not overlap.
When an XML document follows these rules, we say it is well-formed. If a document is not well-formed XML, it is not considered valid XML and cannot be parsed by programs.

Whether it is also a valid document depends on our definition of our document structure.



Because XML elements are nested, they form a tree structure. The more enclosing tags an element has, the lower it appears in the tree structure. Elements with values in them are leaf elements.

It is easy to make a well-formed XML document, but because we are allowed to make our own XML documents, each document can have a different structure. If we want to read and write such documents electronically, we have to be able to define the structure by writing our own rules for our documents. Two formats exist for documenting the structure of XML documents: Document Type Definition and XML Schema.



The document type definition defines every element and every attribute in the document and their relationships. In DTDs, each element is described as either an element with subelements or as a leaf element. The order element has subelements, which are shown in parenthesis. Leaf elements are PCDATA, which stands for parsable character data, meaning simply text that a computer can read.

The attributes of each element are declared using the ATTLIST element. We need one attlist element for each of the elements that has attributes in the document.

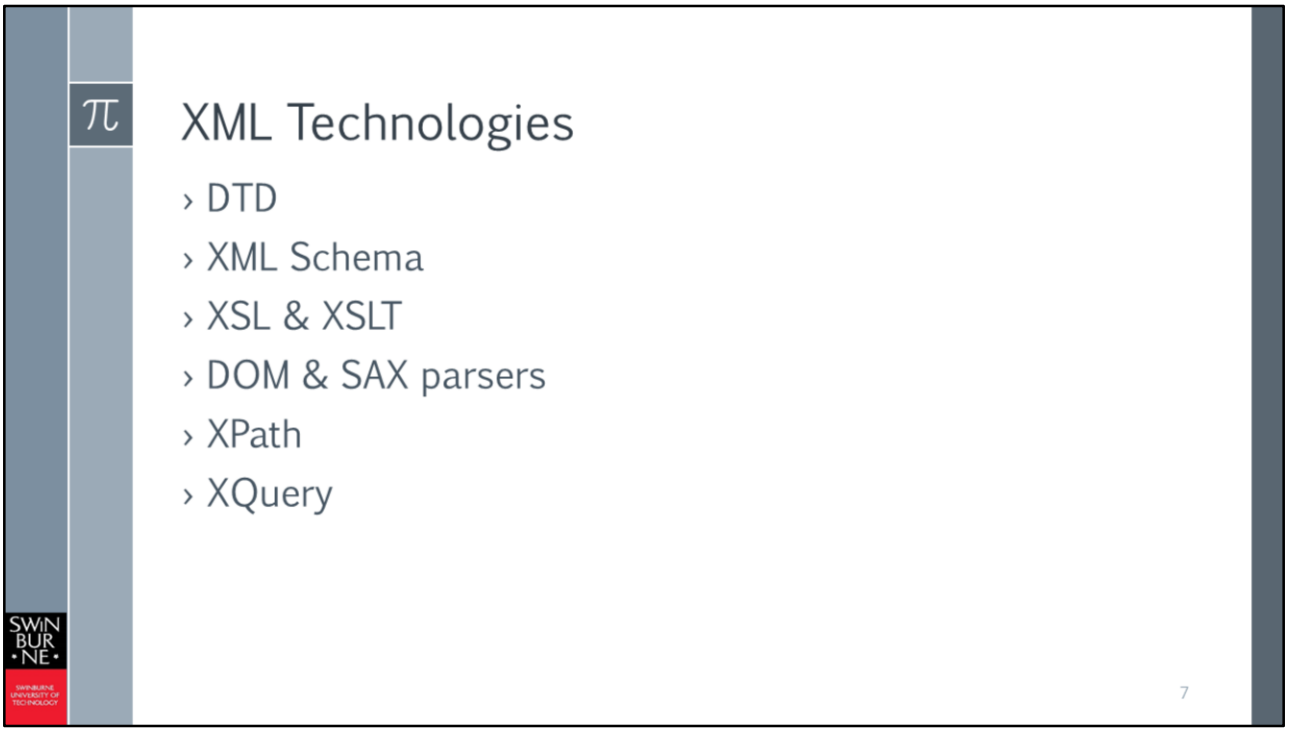
There is an asterisk after order in the top row, where it is declared as a subelement of orders. This means there can be zero or more order elements in the root element called orders. In the second row, the customer, item and delivery elements are declared as subelements of order. The item element can occur one or more times – there is no point having an order if not even a single item has been ordered. You can see that the plus, asterisk and question mark conveniently mean the same as they do in regular expressions.

Note that the format of DTD files is not well-formed XML. Also, DTD has no way of identifying data types. Is quantity a string or an integer?

To solve this problem, XML Schema was created as an XML-compliant definition language for XML documents. Nonetheless, DTD is still being widely used.

XML documents can start with an declaration, which identifies the document type and points to a file that describes the structure of a document. In this case, the DTD or document type description resides with w3.org, the World Wide Web Consortium. Any program that attempts to handle this document can download the description and check whether the document is compliant – meaning valid. To validate an XML document, we need either a DTD or an XML Schema description.

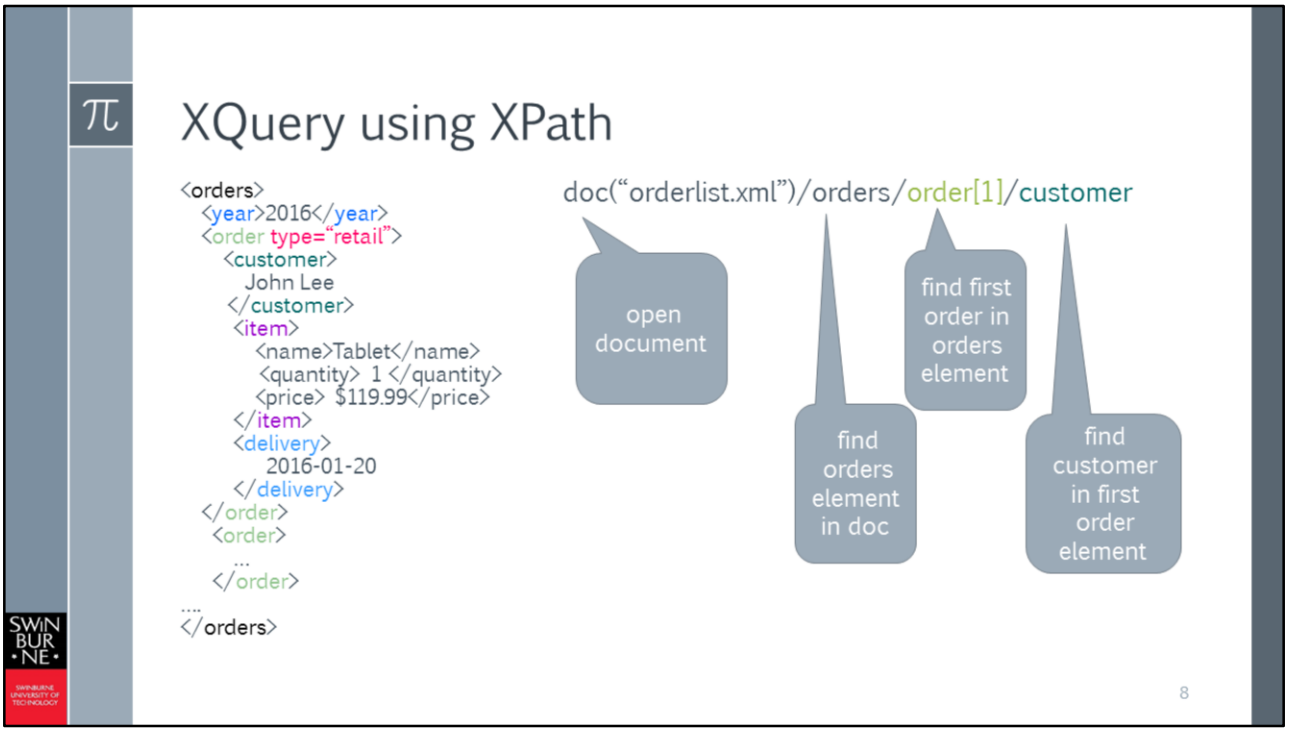
Note that the doctype tag begins with an exclamation mark. This might remind you of the DTD elements of the previous slide. Doctype is a DTD element, not an XML one.
The xmlns attribute determines the namespace. When many different types of xml documents exist, many tags use the same name. Namespaces define the context – is this <header> tag a header in a Microsoft word document or in an xhtml document? The identifier has URL format, but it is more an identifier than a link to a definition.



A great number of technologies were developed to manipulate XML documents. DTDs and XML Schemas are description formats that can define the structure of an XML document. XSL or Extensible Stylesheet Language is the equivalent of CSS – a format that defines how an XML document should be displayed. This is part of the generally accepted strategy of strictly separating the data from its formatting. Using tags, XML tells you what the data means. XSL tells you how it should be shown.

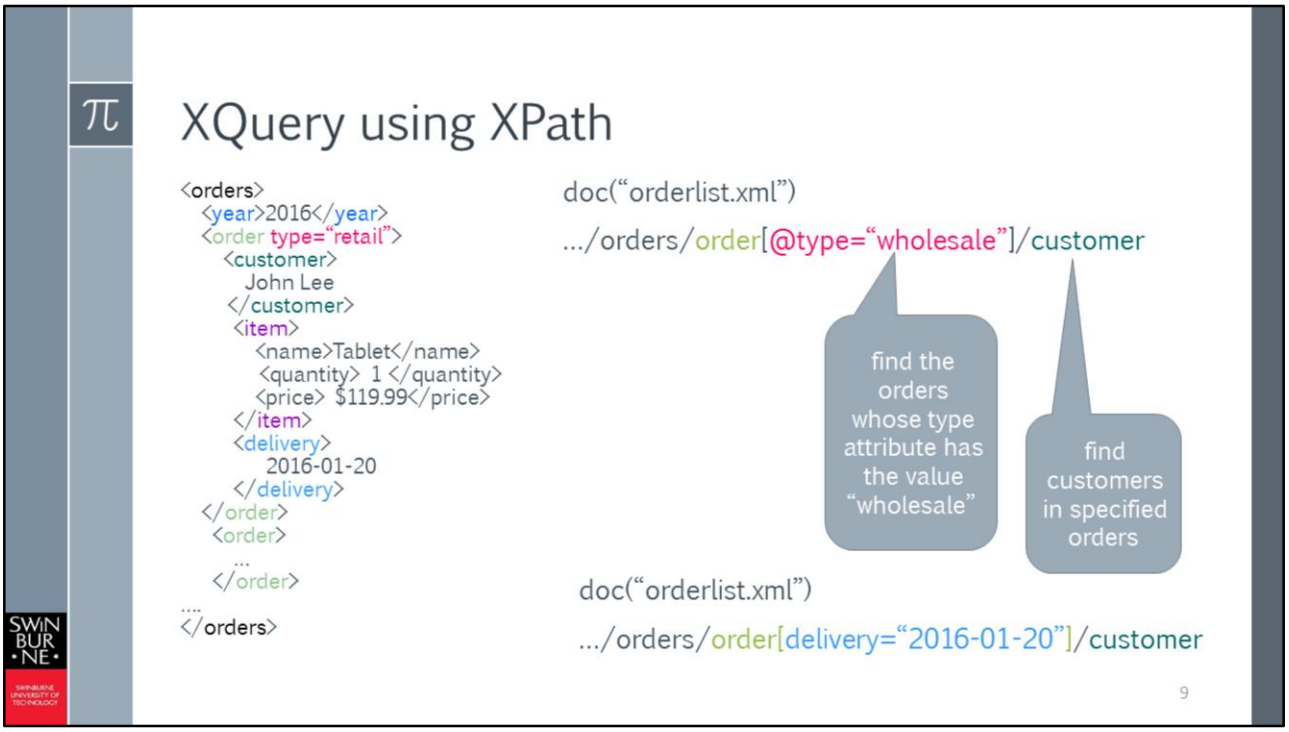
To combine an XML document with XSL formatting instructions, we use XSLT. XSL Transformations not only defines how the document should be rendered but can also change the structure of the document. So you can show the first element of a document last, if you like. This is more powerful than CSS.

XPath provides a way to navigate to particular elements within an XML document. XQuery is a query language for XML that is based on XPath.

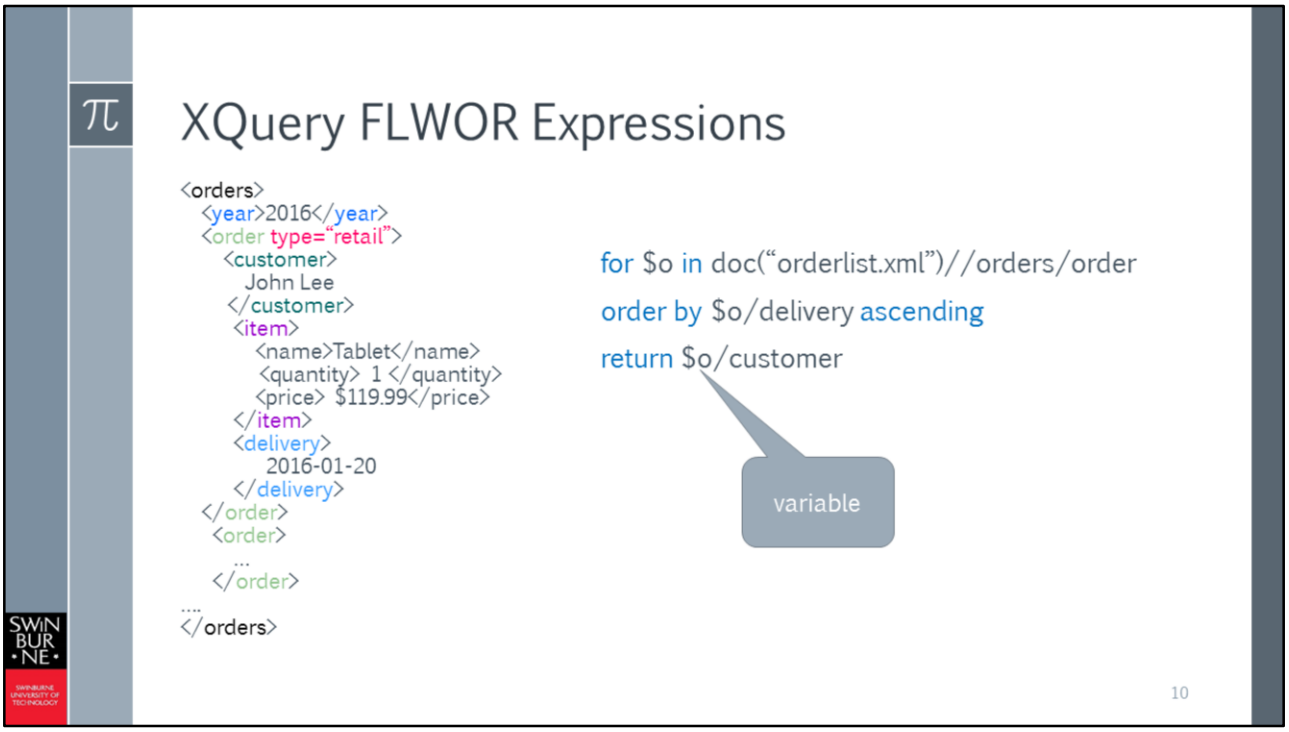


A simple example of an XQuery. Here, the command 'doc' opens the xml document. We assume it has been saved as orderlist.xml.

After the name of the document, the root element of the xml document is mentioned. From there you can navigate along the tree to the desired element by mentioning the name of every element on the way separated by slashes. The order element in the document can have zero or more occurrences. We can include all orders or, as in this example, indicate which one in the sequence we would like. The one in the square brackets means we only want the first element. The actual value we want to see is the customer – the one mentioned last in the sequence of nodes.



In this example, rather than looking up the customer of the first order, we are looking for the customers of all orders whose type attribute is set to wholesale. We can also look for the customer of an order that was delivered on a certain day. The condition – or search criterion - is always shown in square brackets after the element that is being identified. Note the difference between elements and attributes: Attributes have an AT symbol before their names.



Clearly, using the XPath description of the path to an element to query an XML document has its limitations. Advanced XQuery uses FLWOR expressions, pronounced flower. FLWOR stands for FOR, LET, WHERE, ORDER BY and RETURN. FOR means a loop, LET assigns values to variable, WHERE is the condition (or the search criterion or predicate), ORDER BY is a sort command and RETURN returns the outcome.

The example shows how to use XQuery FLWOR to loop through all orders to sort them. Variables start with a $ sign and the $o variable is assigned to an order at a time. Having sorted all orders by the delivery date, we return only the customer names. This lists the customer names in order of the dates of their deliveries.

XQuery FLWOR Expressions, continued

```
<orders>
  <year>2016</year>
  <order type="retail">
    <customer>
      John Lee
    </customer>
    <item>
      <name>Tablet</name>
      <quantity> 1 </quantity>
      <price> 119.99</price>
    </item>
    <item>
      <name>PC</name>
      <quantity> 1 </quantity>
      <price> 450.99</price>
    </item>

    <delivery>
     2016-01-20
    </delivery>
  </order>
....
</orders>
```

```
for $o in doc("orderlist.xml")/orders/order
let $i := $o/item
let $total := sum($i/quantity*$i/price)
return
<purchase>
        <name> {$o/customer/text()} </name>
        <total> {$total} </total>
</purchase>
```

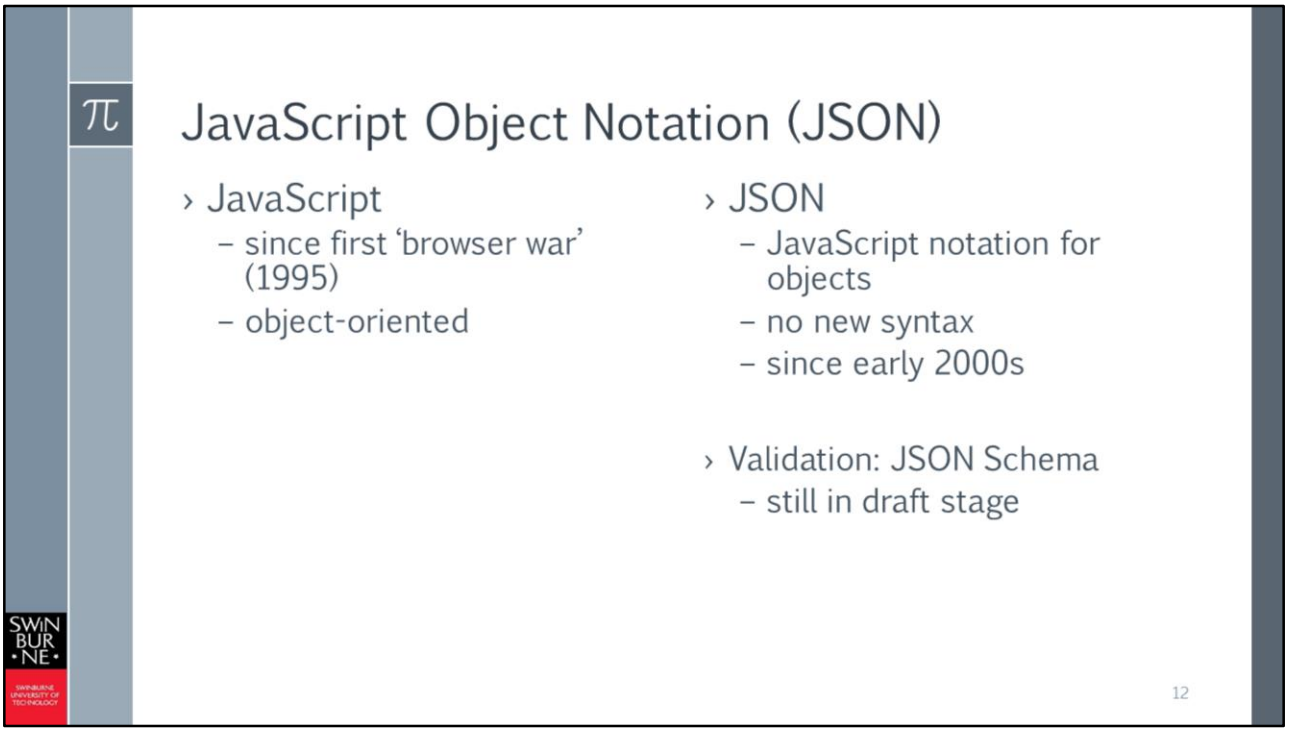
```
<purchase>
        <name> John Lee </name>
        <total> 119.99 </total>
</purchase>
<purchase>
        <name> Adam Connor </name>
        <total> 981.98 </total>
</purchase>
```

11

Here is a more complex example. The goal is to list all customers and their order totals. We assume that there is mostly more than one item in each order.

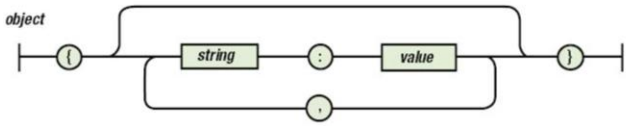
To list the customers, per order, we have to loop through the orders – this is what the for loop does, capturing each order element in the variable $o. The second line assigns the item elements within the current order element to the variable i.

In the third line, the sum of the quantities multiplied by the prices are assigned to the total variable. We return a new xml format for the outcome. The customer first and last names are now the values of the new name element. Note the text() function. The subelement customer of the o variable denotes the entire element with the customer tags and the value. To include only the value, we have to call the text function on the customer element.



In 1995, Netscape (the company behind Netscape Navigator) wanted to add dynamic behaviour to web pages within its browser. JavaScript was developed as an object-oriented scripting language to be used in browsers (client-side) to create behaviour such as changing the colour of a button when a mouse was pointed at it or expanding menus. Although XML was being used in many web technologies for data interchange, many people found that XML is unnecessarily bulky. Having start and end tags adds significantly to the number of characters needed to represent an object.
In the early 2000s some people started using JavaScript notation for objects to represent data while it is being sent over the network to another computer. JSON came into existence. Since JSON emerged without specific development, a standard was adopted only in 2009, and now, two almost identical standards exist. Only one of the standards specifies that a JSON document has to have a single root element. Since people just started using JSON as a data format, no supporting technologies like document description formats for validation were developed initially. A draft Schema is only being developed now.

JSON Object



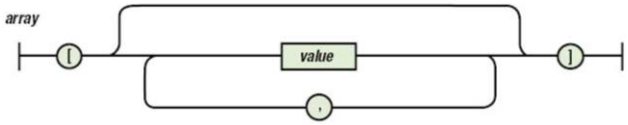
JavaScript
var person = {name:"John", age:15}

JSON
{name:"John", age:15}

13

JavaScript specifications use so-called railroad diagrams to represent the notation. When representing objects, we encase them in curly braces. The upper line of the diagram represents the case where the object is empty – the string:value pair is skipped. The middle line represents a single key-value pair. The lower line represents repetition and shows that if the key-value pair repeats, the pairs are separated by a comma.
Clearly, the same notation is used in JavaScript. In the example, an object with a name of John and an age of 15 is assigned to the variable person.

JSON Array

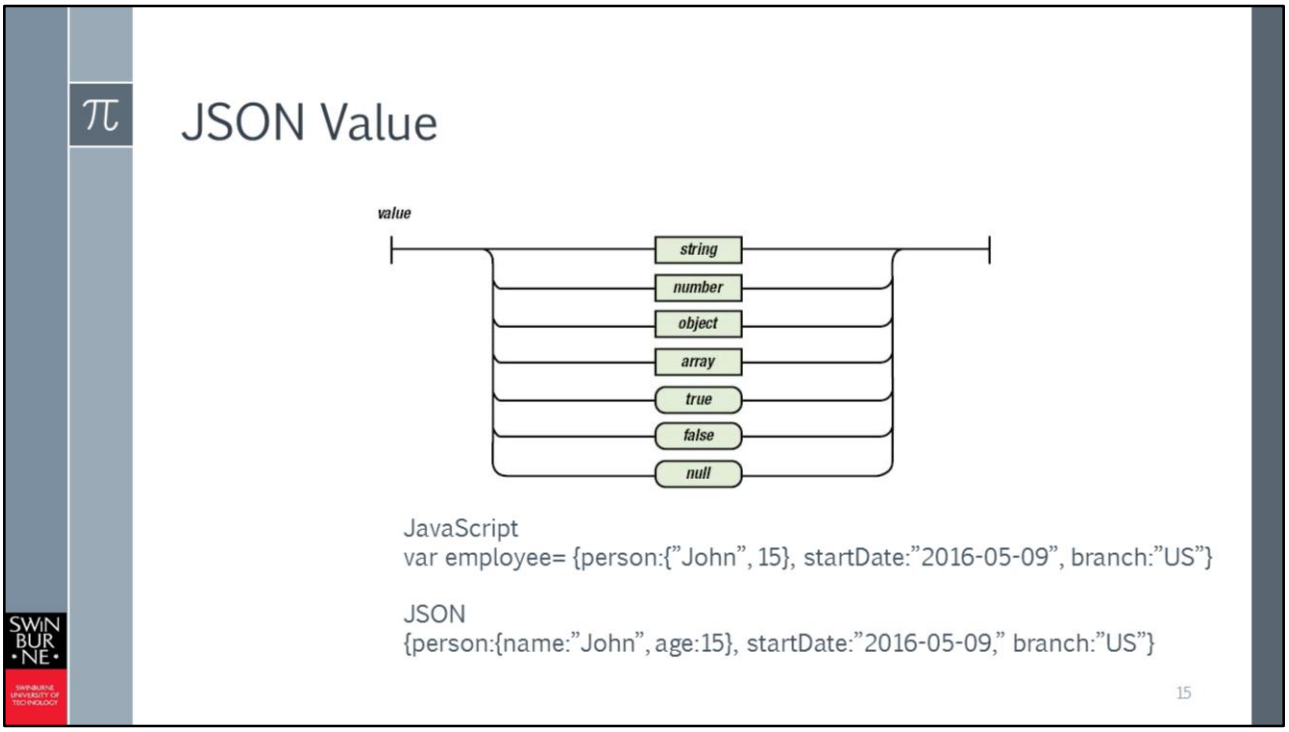


JavaScript
var person = ["John", 15]

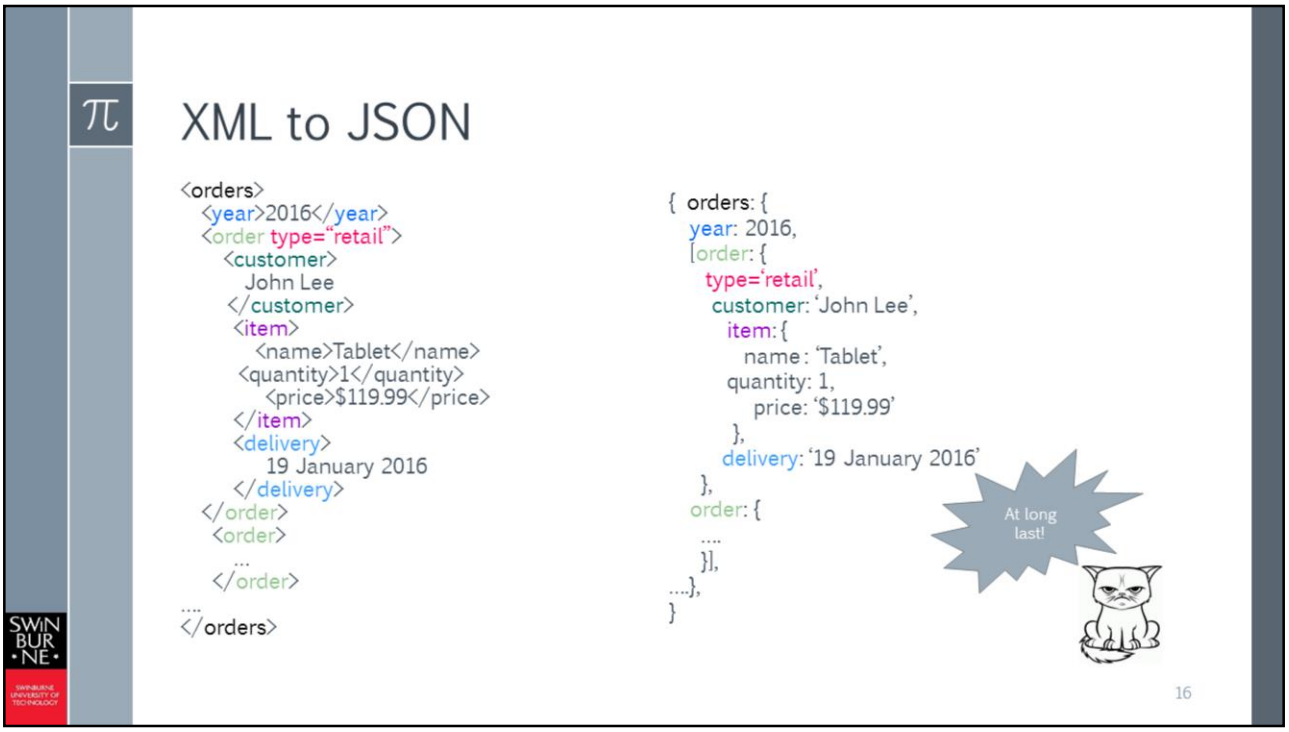
JSON
[name:"John", age:15]

14

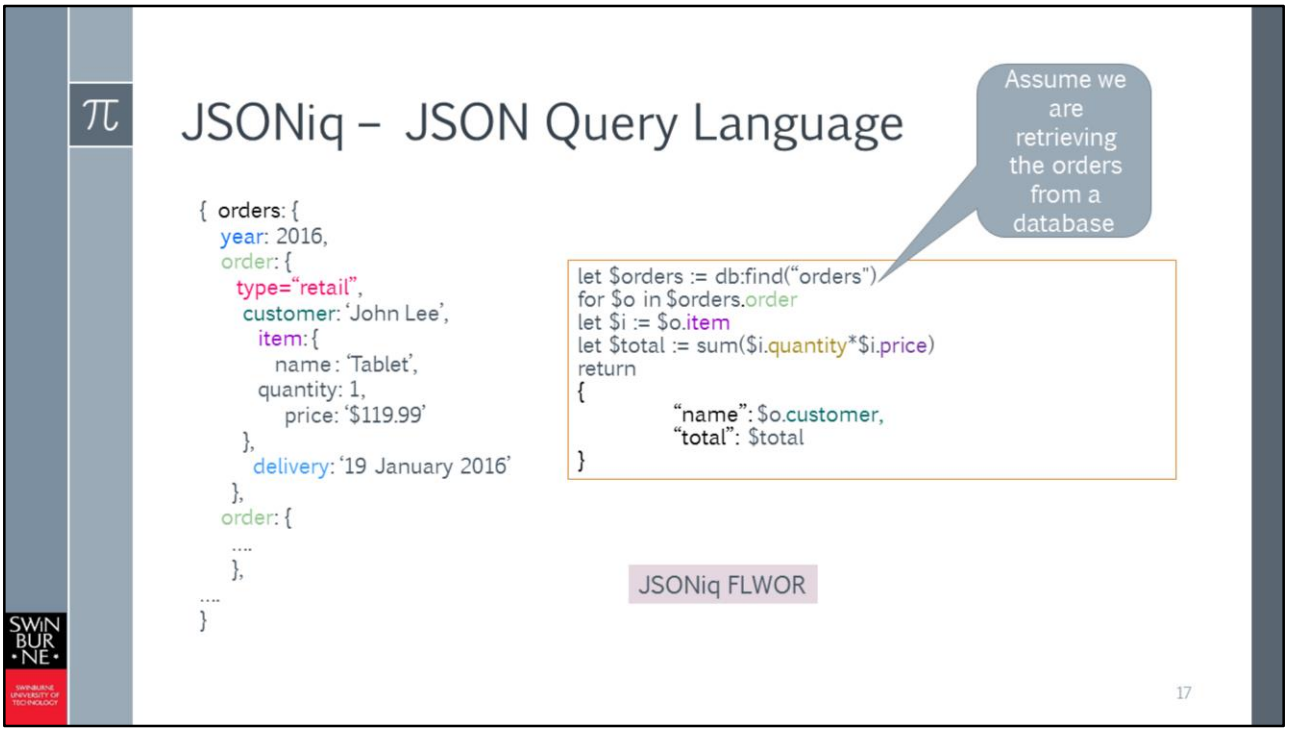
The railroad diagram shows that a JSON array is represented encased in square brackets. Again, the upper line represents skipping any content. The middle line shows that a single value can be added to the array. The lower line shows that if there are several of these values, they are separated by commas. Note that the members of arrays do not have to be of the same type. Here we have a string and a number as part of an array. The difference between an array and an object is that objects have key-value pairs and arrays only have values.



In the object and array diagrams, the core ingredient is the value. What types of values can we have inside an object or an array? The type can be a string, a number, an object, an array, a Boolean or a null, as shown in this railroad diagram. As in object-oriented languages, objects can contain other objects but also 'primitive types': numbers and character data, Booleans.
In the example, we have one object – the person John - and two primitive types as part of an object.
Note that strings have to be written in double quotes.

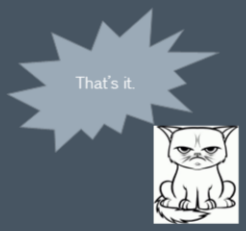


Here we have the XML and JSON notations of the same document. The orders object contains a string value with the key year and several order objects. The order object contains item objects and other key-value pairs. An interesting detail is the type key-value pair: in XML, this was modelled as an attribute. In JSON, there are no attributes; it is just another key-value pair. In XML, one of the tricky questions always was whether to represent a detail as an element or an attribute. This problem does not exist in JSON.



JSONiq has the same FLWOR constructs as XQuery. You can see that the notation is the same, but the slashes that separate the levels of the tree in the notation have been replaced by dots. The output is naturally in JSON form rather than XML. JSONiq has also many other features that are inspired from SQL.

Summary



- The most prevalent semi-structured data formats are XML and JSON.
- XML is more verbose and widely used in web-based technologies developed between 1995 and 2005.
- XML comes with many supporting technologies such as DTDs and XML Schema that enable the user to enforce structure, and every major programming languages has libraries for parsing and handling XML.
- JSON is a more recent format that replaces XML providing a similar tree structure. NoSQL databases often use JSON
- JSON has its own query language JSONiq, the equivalent of XQuery in the XML area.

18

Here are the most important points discussed in this module. You may want to stop the recording to have a read through them. When you are ready, start the quiz about this module.