Capgemini

XML Schema Fundamentals

Welcome

academy

Version 1.0

Capgemini

XML Schema Fundamentals

Intro

academy

Version 1.0

INTRODUCTIONS

- Who am I?
- Who are you?
- Agenda
- Agreements

academy

TRAINER

Marlies Hoefkens

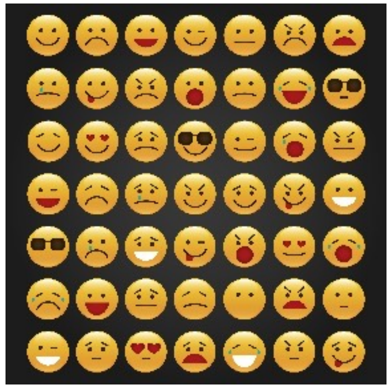
- Amersfoort

- Hobbies

- Specialised in IT and Agile training

WHO ARE YOU?

- Your name and role in your organisation
- Your experience with XML, HTML and programming
- What is your goal this training and what do you to achieve?



GOALS OF THIS TRAINING?

- End of this training:
  - Understand the fundamentals of XML, including its syntax and structure.
  - Learn how to define the structure of an XML document using Document Type Definitions (DTD) and XML Schema Definition (XSD).
  - Explore XPath and XQuery as query languages for navigating and querying XML documents.
  - Gain proficiency in transforming XML data into various formats using XSLT.
  - Understand XML parsing and processing techniques using DOM and SAX.
  - Acquire best practices for creating well-formed and valid XML documents.
  - Develop an awareness of performance considerations and security aspects related to XML.
  - Empower participants to confidently create, process, and exchange XML data.

academy

AGREEMENTS ON…

Laptops

Breaks

Lunch

Phones

Attendeeslist

Questions

XML Schema Fundamentals

Introduction

academy

XML

•XML (Extensible Markup Language) as a popular format for representing structured data.

•Benefits of using XML Schema: validation, documentation, and interoperability.

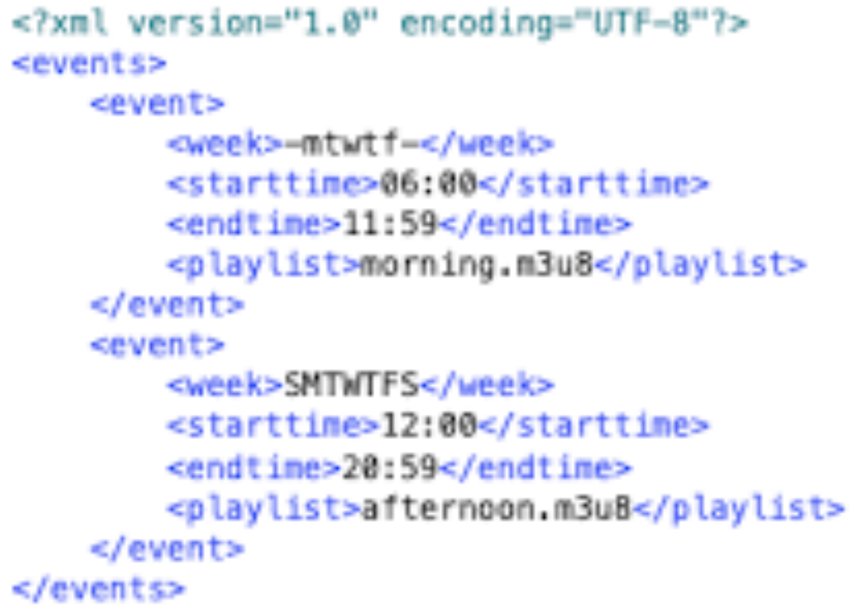
•Element: XML document's basic building block for data or structure.

•Namespace: Avoids naming conflicts via unique identifiers for elements and attributes.

•Inheritance: Defines new elements by inheriting structure and attributes.

WHAT IS XML?

- XML is a language that allows us to represent structured data in a machine-readable and human-readable format.
- Unlike HTML, which is primarily used for presenting data on the web, XML focuses on data representation and structure.
- XML provides a flexible way to define custom tags and structures that suit the specific requirements of the data being represented.



```
<?xml version="1.0" encoding="UTF-8"?>
<events>
    <event>
        <week>-mtwtf-</week>
        <starttime>06:00</starttime>
        <endtime>11:59</endtime>
        <playlist>morning.m3u8</playlist>
    </event>
    <event>
        <week>SMTWTFS</week>
        <starttime>12:00</starttime>
        <endtime>20:59</endtime>
        <playlist>afternoon.m3u8</playlist>
    </event>
</events>
```

WHY XML?

- Data Interchange: Its standardized structure and human-readable format make it suitable for exchanging data between different applications and organizations.
- Web Development: XML plays a significant role in web development, particularly in data representation and communication (SOAP, REST, API)
- Document Storage and Management: XML provides a structured and hierarchical approach to storing and managing documents. It allows for the organization and categorization of data within the documents, making it easier to search, retrieve, and manipulate information.
- Configuration Files: XML is often used for storing configuration settings for software applications and systems
- Data Modeling and Schema Definition: XML provides a flexible way to define data models and structures through Document Type Definitions (DTD) or XML Schema Definition (XSD). It allows for the validation and enforcement of data integrity and rules, ensuring data consistency and interoperability.
- Cross-Platform Compatibility: XML is platform-independent and can be processed and interpreted by various programming languages and systems.

XML Schema Fundamentals

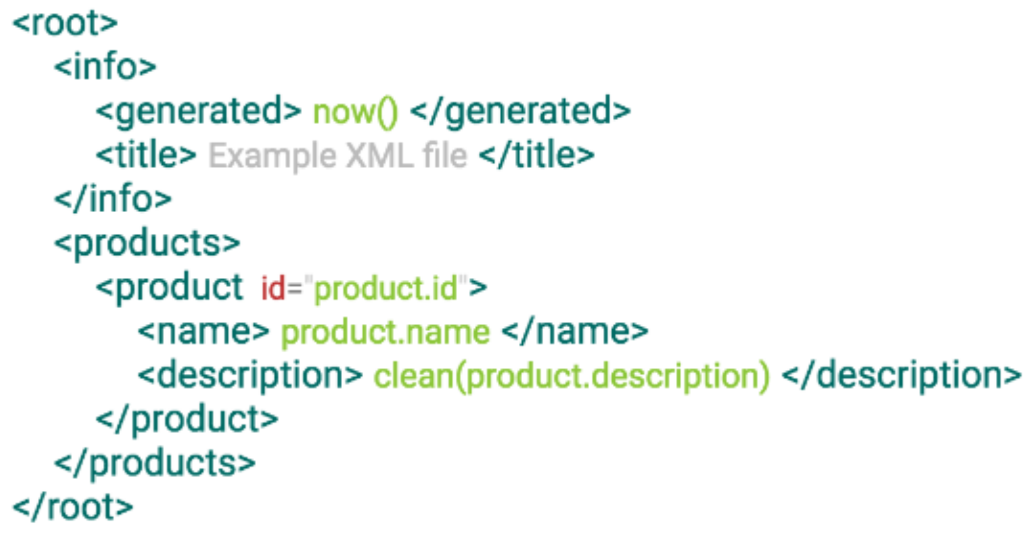
Syntax, structure and rules

academy

XML SYNTAX AND STRUCTURE

XML has a specific syntax that must be followed. It consists of opening and closing tags to define elements.

- Tags are enclosed in angle brackets (< >), and elements are organized in a hierarchical structure.
- Each opening tag must have a corresponding closing tag, and elements can be nested within each other to create a tree-like structure.



```
<root>
  <info>
    <generated> now() </generated>
    <title> Example XML file </title>
  </info>
  <products>
    <product id="product.id">
      <name> product.name </name>
      <description> clean(product.description) </description>
    </product>
  </products>
</root>
```

XML SYNTAX AND RULES

XML Syntax:

•XML uses tags

•Opening and closing tags enclose content

•Tags are case-sensitive

•Elements can have attributes enclosed within the opening tag.

•Text content is placed between opening and closing tags.

•Empty elements can be self-closed

XML Rules:

•Must have a single root element.

•Tags must be properly nested

•Attribute values must be enclosed in quotes (single or double).

•Reserved characters (<, >, &, ', ")

•XML is Unicode-based

•Comments can be added using <!-- -->.

XML ELEMENTS VS. ATTRIBUTES

Attributes

```
<product id="1234">
<name>Widget A</name>

<category>Electronics</category>
<price
currency="USD">49.99</price>
</product>
```

The <product> element has an attribute id with a value of 1234, representing the unique identifier for the product.

Elements

```
<product>
<id>5678</id>
<name>Widget B</name>
<category>Home &
Kitchen</category>
<price>
<amount>29.99</amount>
<currency>USD</currency>
</price>
</product>
```

The <product> element does not have an attribute for the ID. Instead, it uses an <id> element to represent the product ID, with a value of 5678.

EXAMPLE

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bank>
  <account type="savings">

<accountNumber>1234567890</accountNumber>
    <customerName>John Doe</customerName>
    <balance>5000.00</balance>
  </account>
```

```xml
 <account type="checking">

<accountNumber>9876543210</accountNumb
er>
    <customerName>Jane
Smith</customerName>
    <balance>2500.00</balance>
  </account>
</bank>
```

EXERCISES

- You can use any code editor to create an XML(.xml) file
  - Notepad(++)
  - Save as .xml
  - View in browser
- Online sandbox for validating: https://jsonformatter.org/xml-formatter
- https://www.w3schools.com/xml/tryxslt.asp?xmlfile=cdcatalog&xsltfile=cdcatalog

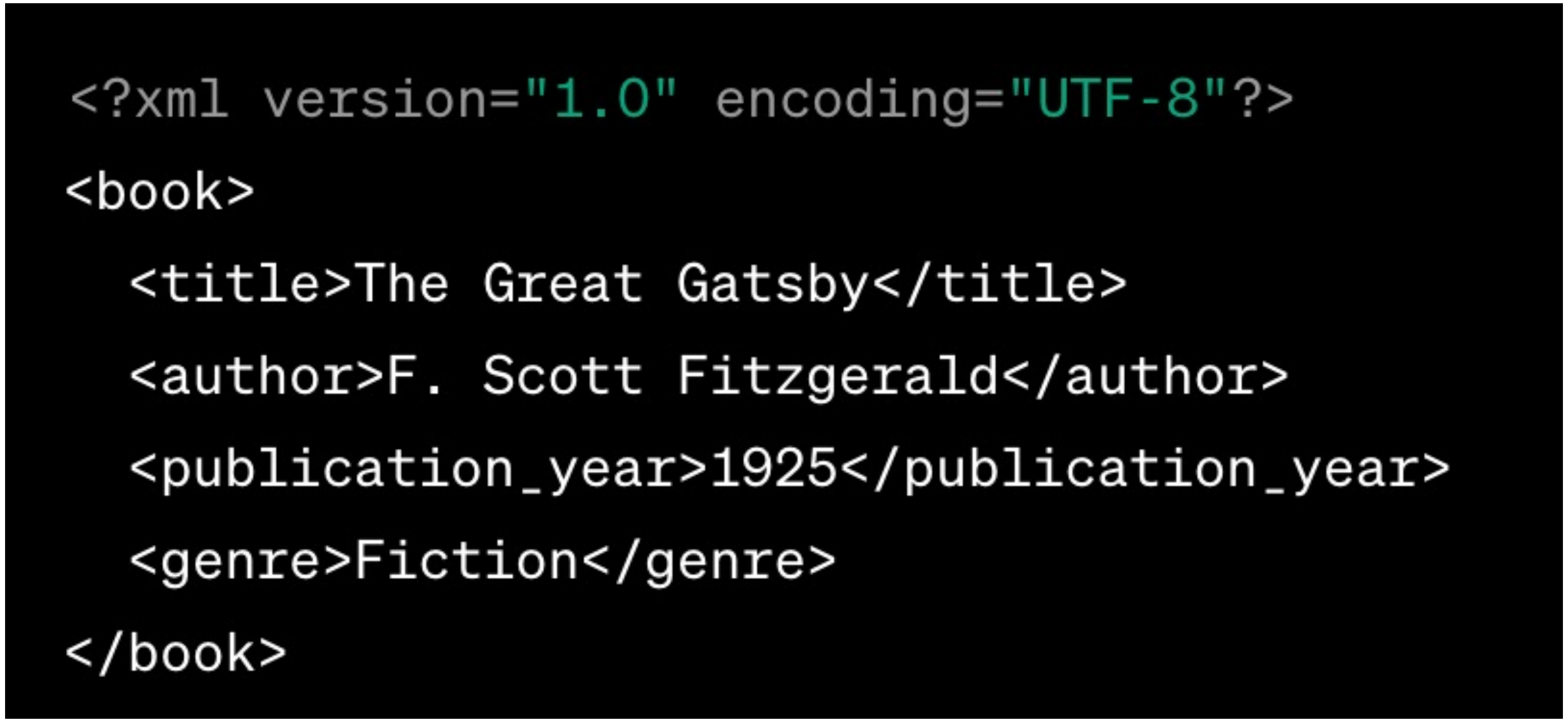
EXERCISE 1

Consider the following scenario where you need to represent information about books in XML format. Design an XML structure that highlights the difference between XML elements and attributes.

Requirements:

- Include the book's title, author(s), publication year, and genre.

EXERCISE 1



```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <title>The Great Gatsby</title>
  <author>F. Scott Fitzgerald</author>
  <publication_year>1925</publication_year>
  <genre>Fiction</genre>
</book>
```

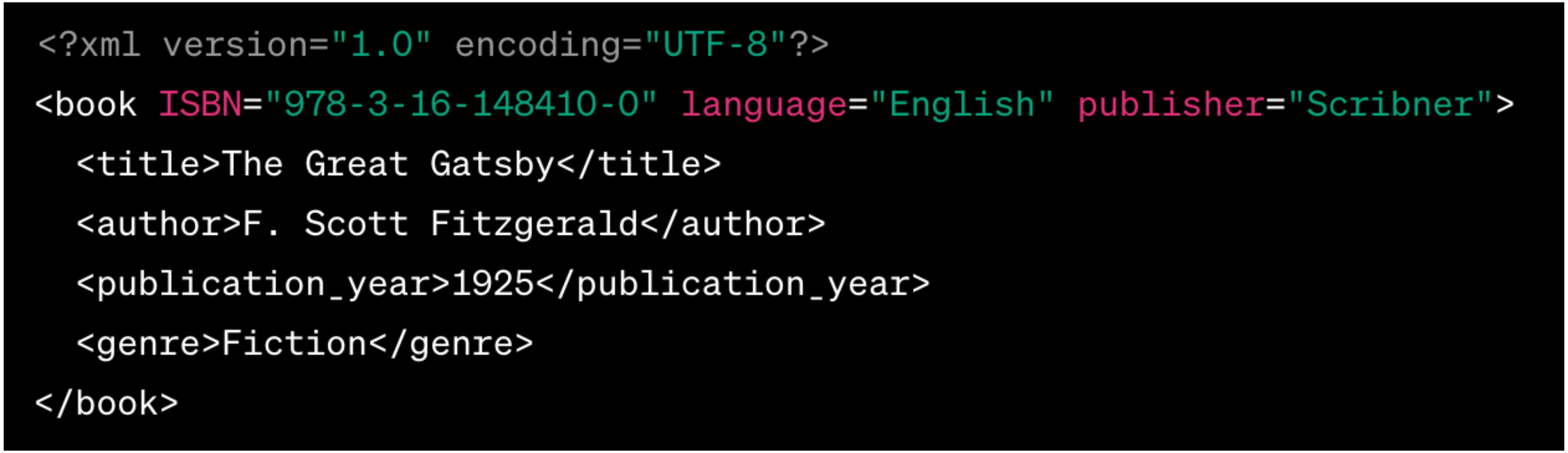
EXERCISE 2

Add attributes to XML elements.

Instructions:

- Take the XML document from Exercise 1 and add attributes to the elements (ISBN).
- For example, add attributes like language, and publisher to the book element.
- Update the XML document accordingly and validate its structure.
- Choose at least one piece of information that is better suited as an attribute rather than an element.
- Provide multiple examples of books to demonstrate the use of both elements and attributes.

EXERCISE 2



```xml
<?xml version="1.0" encoding="UTF-8"?>
<book ISBN="978-3-16-148410-0" language="English" publisher="Scribner">
  <title>The Great Gatsby</title>
  <author>F. Scott Fitzgerald</author>
  <publication_year>1925</publication_year>
  <genre>Fiction</genre>
</book>
```

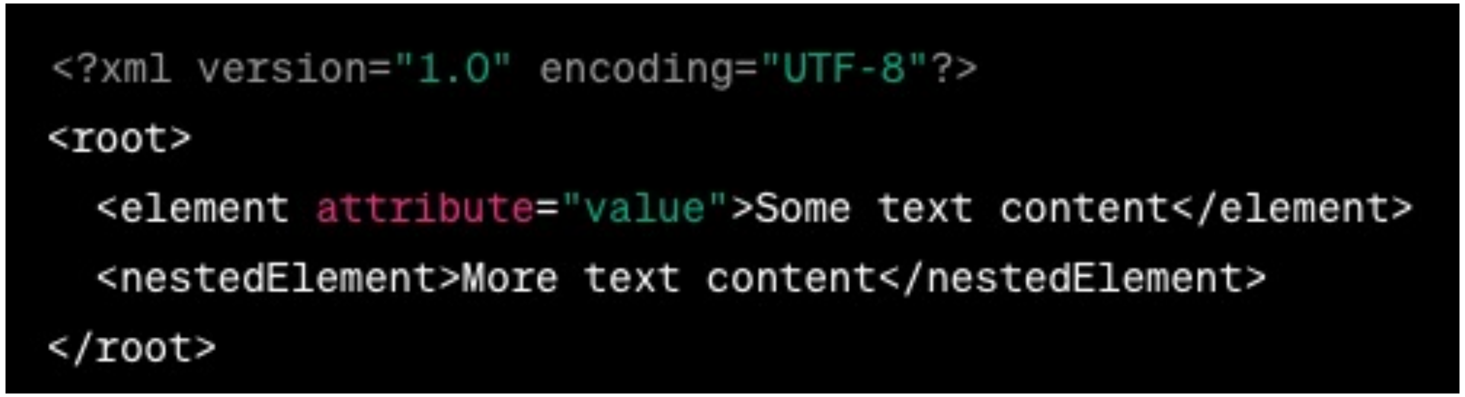
XML Schema Fundamentals

Nesting

academy

XML NESTING

- Placing an element in an element
- Hierarchy
- "Parent" and "child" element
- Complex data structures
- Called wrapper or container



```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <element attribute="value">Some text content</element>
  <nestedElement>More text content</nestedElement>
</root>
```

EXERCISE 3

Nest elements within an XML document.

Instructions:

- Create an XML document representing a recipe for a dish.
- Include elements for recipe name, ingredients, and instructions.
- Nest the ingredient elements within the ingredients element and the instruction elements within the instructions element.

EXERCISE 3

```xml
<?xml version="1.0" encoding="UTF-8"?>
<recipe>
  <name>Chocolate Chip Cookies</name>
  <ingredients>
    <ingredient>1 cup butter</ingredient>
    <ingredient>1 cup sugar</ingredient>
    <ingredient>2 cups flour</ingredient>
    <!-- Additional ingredients here -->
  </ingredients>
  <instructions>
    <instruction>Preheat the oven to 350°F.</instruction>
    <instruction>Cream the butter and sugar together.</instruction>
    <instruction>Add flour gradually and mix well.</instruction>
    <!-- Additional instructions here -->
  </instructions>
</recipe>
```

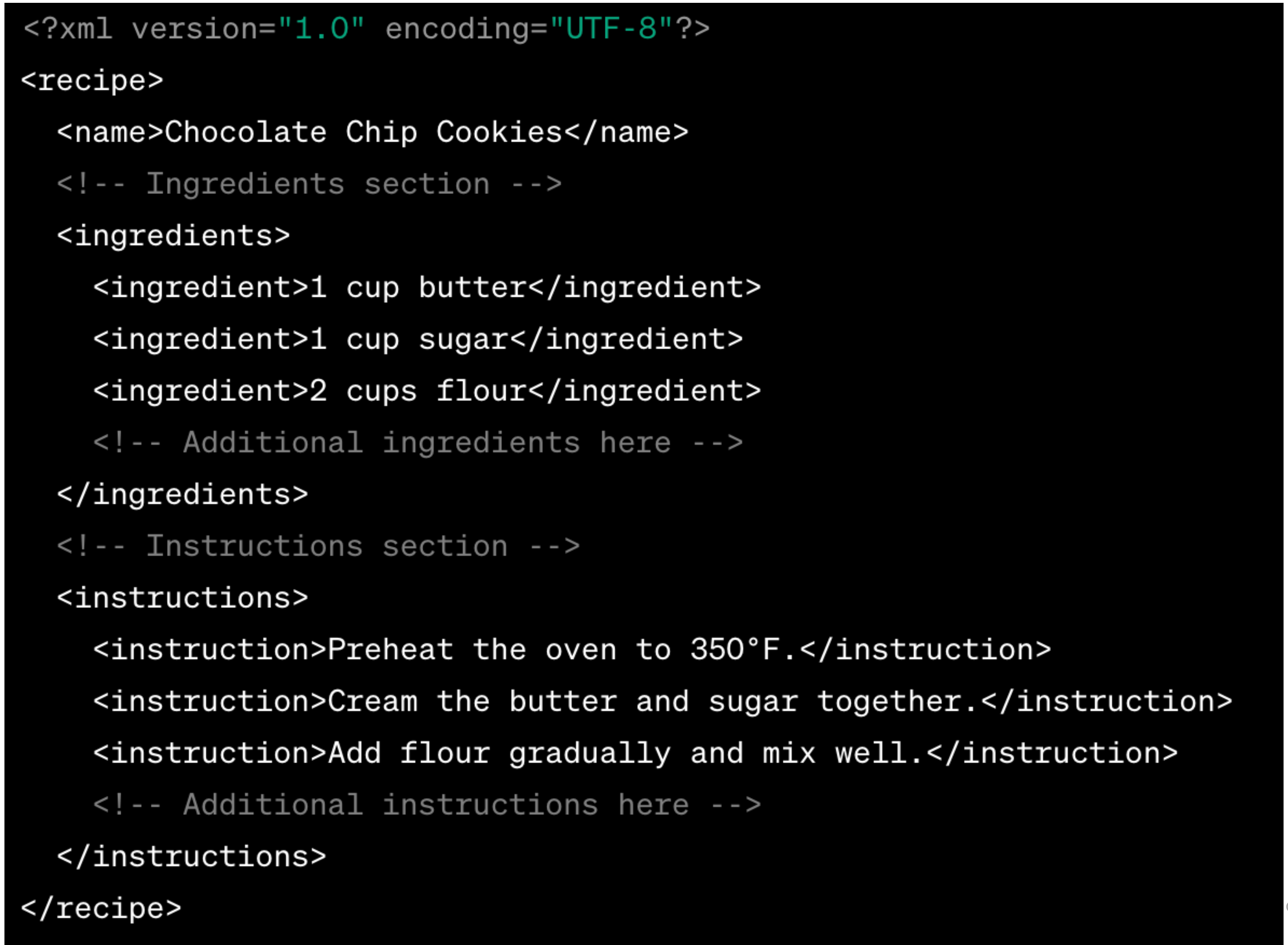
EXERCISE 4

Use comments in an XML document.

Instructions:

- Take the XML document from Exercise 3 and add comments to describe each element or provide additional information.
- Use comments to explain the purpose of elements, provide clarifications, or highlight any important details.
- Validate the updated XML document and verify that the comments are ignored during parsing.

EXERCISE 4



```xml
<?xml version="1.0" encoding="UTF-8"?>
<recipe>
  <name>Chocolate Chip Cookies</name>
  <!-- Ingredients section -->
  <ingredients>
    <ingredient>1 cup butter</ingredient>
    <ingredient>1 cup sugar</ingredient>
    <ingredient>2 cups flour</ingredient>
    <!-- Additional ingredients here -->
  </ingredients>
  <!-- Instructions section -->
  <instructions>
    <instruction>Preheat the oven to 350°F.</instruction>
    <instruction>Cream the butter and sugar together.</instruction>
    <instruction>Add flour gradually and mix well.</instruction>
    <!-- Additional instructions here -->
  </instructions>
</recipe>
```

XML Schema Fundamentals

Data types

academy

XML DATA TYPES

XML Schema provides data types to define content in XML elements and attributes.

Built-in Data Types: String, Numeric (integers, decimals, floats, doubles), Boolean, Date and Time, Binary, AnyURI.

Derived Data Types: Inherit from built-in types with added constraints or modifications

Custom Data Types: Create types based on specific requirements by combining built-in or derived types.

Type Restrictions and Facets: Use restrictions and facets to further constrain and define data types)

Benefits of Data Types: Enforce data integrity, Facilitate interoperability, Enhance data validation, Improve documentation.

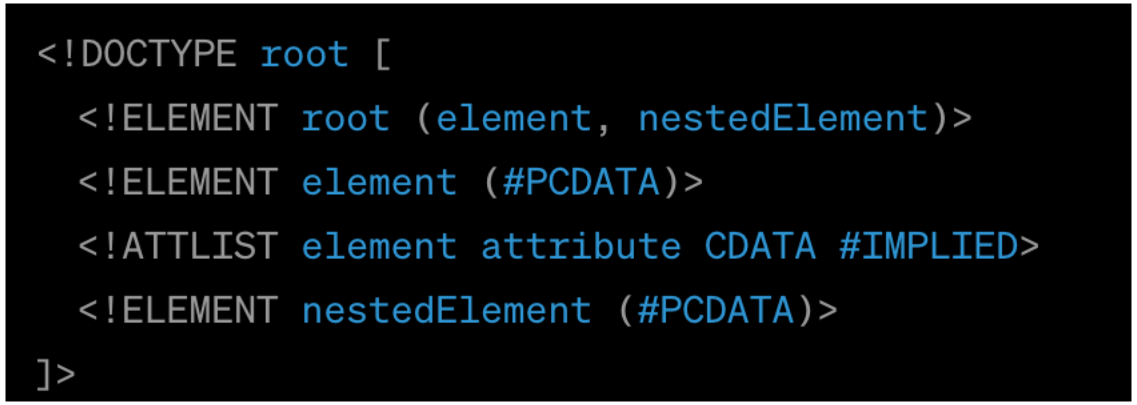
XML Schema Fundamentals

Schema models

academy

DOCUMENT TYPE DEFINITION (DTD)

1. DTD is an older and simpler method for defining the structure of an XML document.
2. DTD uses a set of declarations to define elements, attributes, entities, and their relationships.
3. DTD declarations are placed within the <!DOCTYPE> declaration at the beginning of an XML document.
4. DTD supports basic data types and limited validation capabilities.



```
<!DOCTYPE root [
  <!ELEMENT root (element, nestedElement)>
  <!ELEMENT element (#PCDATA)>
  <!ATTLIST element attribute CDATA #IMPLIED>
  <!ELEMENT nestedElement (#PCDATA)>
]>
```

XML SCHEMA DEFINITION (XSD):

1. XSD is a more powerful and flexible method for defining XML document structures.
2. XSD uses its own XML-based syntax to define elements, attributes, complex types, data types, and their relationships.
3. XSD supports a wide range of data types, including built-in types and user-defined types.
4. XSD provides advanced validation capabilities, such as data type validation, cardinality constraints, and more.

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="element" type="xs:string">
          <xs:complexType>
            <xs:attribute name="attribute" type="xs:string" />
          </xs:complexType>
        </xs:element>
        <xs:element name="nestedElement" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

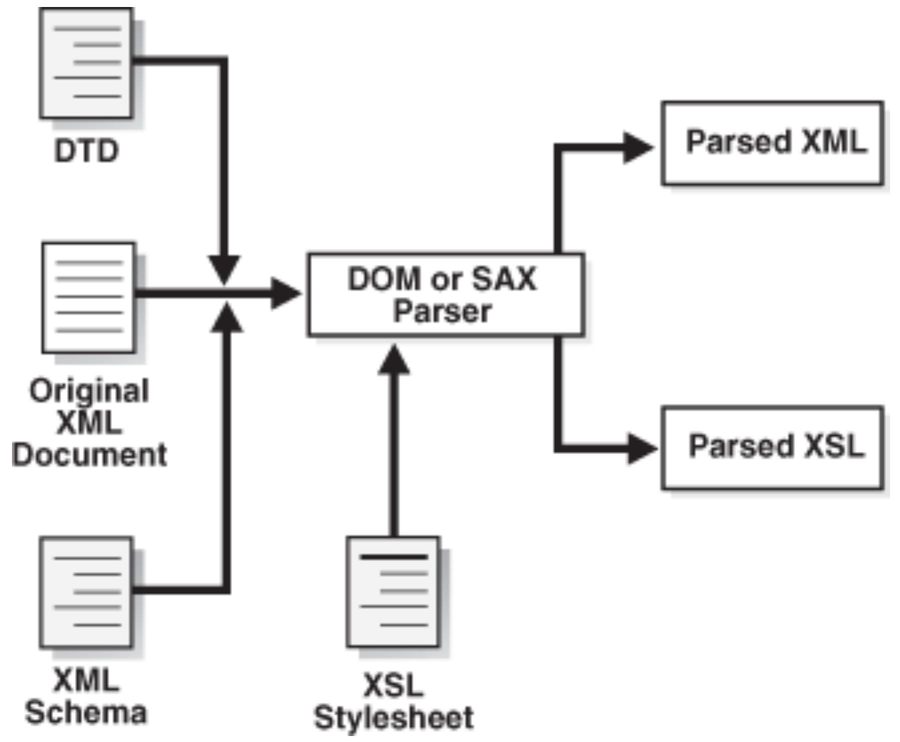
XML Schema Fundamentals

Parsing

academy

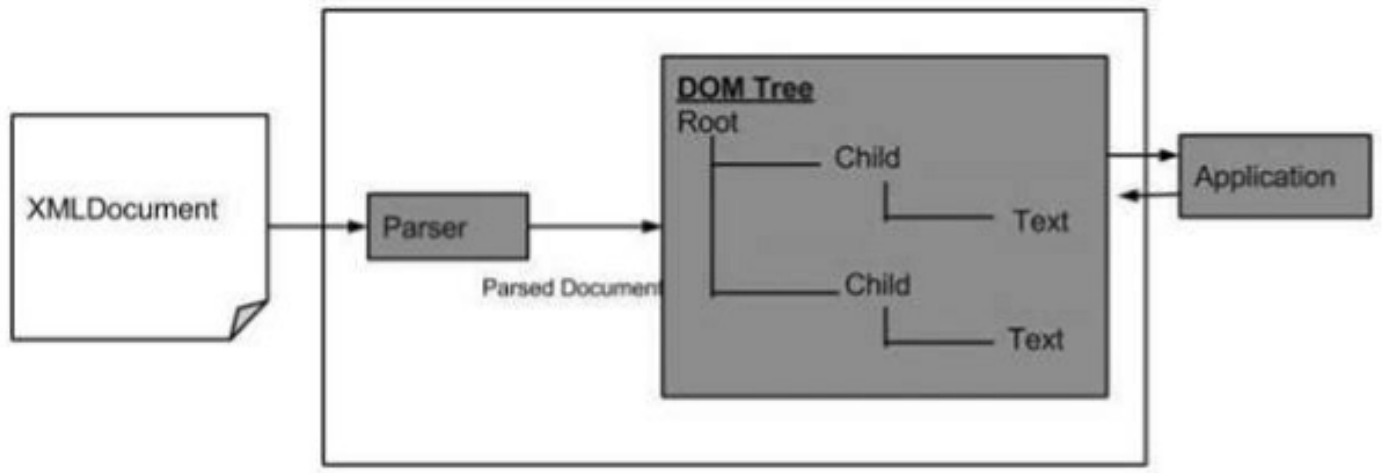
XML PARSING

Parsing XML refers to the process of analysing the structure and content of an XML document to extract information and perform specific operations on the data. XML parsing is commonly performed by software applications or libraries known as XML parsers.



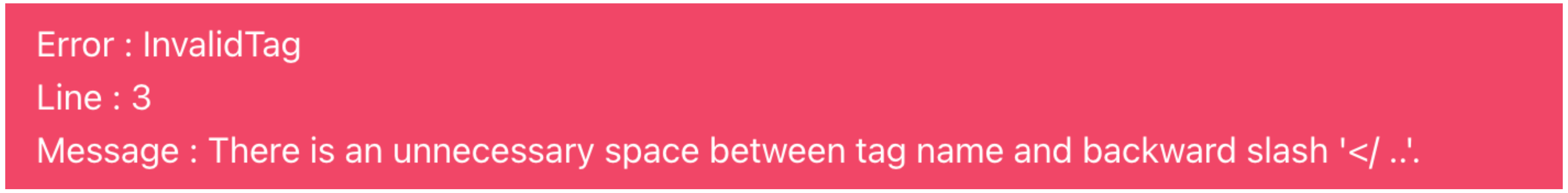
XML PARSING METHODS

1. DOM (Document Object Model): DOM parsing creates a tree-like representation of the entire XML document in memory. It allows easy traversal and manipulation of the XML structure but may consume more memory for large documents.
2. SAX (Simple API for XML): SAX parsing is an event-driven approach where the XML parser notifies the application about various events, such as start tags, end tags, and content. It is efficient for processing large XML documents but may require more complex code for data extraction.
3. StAX (Streaming API for XML): StAX parsing provides a streaming model where the application pulls the XML data as needed. It combines the advantages of DOM and SAX parsing, providing a simpler programming model while minimizing memory usage.



XML PARSING ERROR HANDLING

1. XML parsing can encounter errors such as malformed XML, missing elements, or invalid data.
2. Proper error handling is crucial to handle exceptions, validate XML against schemas or DTDs, and provide meaningful error messages to users.



```
Error : InvalidTag
Line : 3
Message : There is an unnecessary space between tag name and backward slash '</ ..'.
```

XML PARSING DATA VALIDATION

1. XML parsers can perform data validation against an XML Schema Definition (XSD) or a Document Type Definition (DTD).
2. Validation ensures that the XML document adheres to a predefined structure, data types, and constraints, enhancing data integrity and reliability.

XML PARSING SECURITY CONSIDERATIONS

1. XML parsing may involve potential security risks, such as entity expansion attacks or XML External Entity (XXE) attacks.
2. Implement security measures like disabling external entity resolution, employing input validation, and utilizing secure parsers to mitigate these risks.

XML PARSING PERFORMANCE OPTIMIZATION

1. XML parsing can impact the performance of applications, especially when dealing with large XML documents.
2. Techniques such as streaming parsing, using efficient parsers, or employing caching mechanisms can help optimize XML processing.

EXERCISE 5

Go to: https://jsonformatter.org/xml-editor

Instructions:

- Take the XML document from Exercise 3, paste and try to make an error.
- View in tree view.
- What do you observe?

DISPLAYING XML WITH XSLT

- XSLT (eXtensible Stylesheet Language Transformations) is the recommended style sheet language for XML.
- XSLT is far more sophisticated than CSS. With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.
- XSLT uses XPath to find information in an XML document.

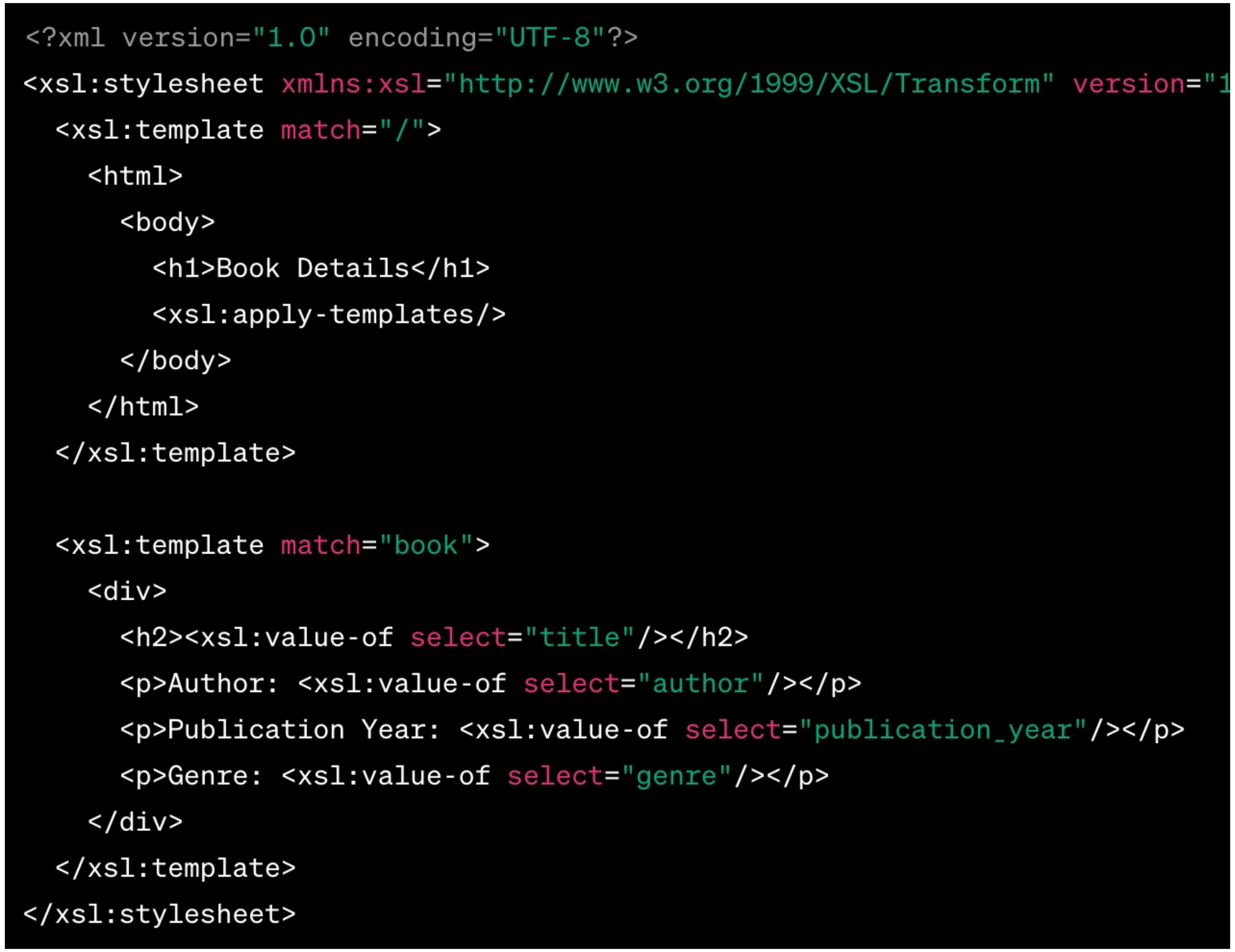
EXERCISE 6

Transform XML to HTML using XSLT. Use W3schools.com!

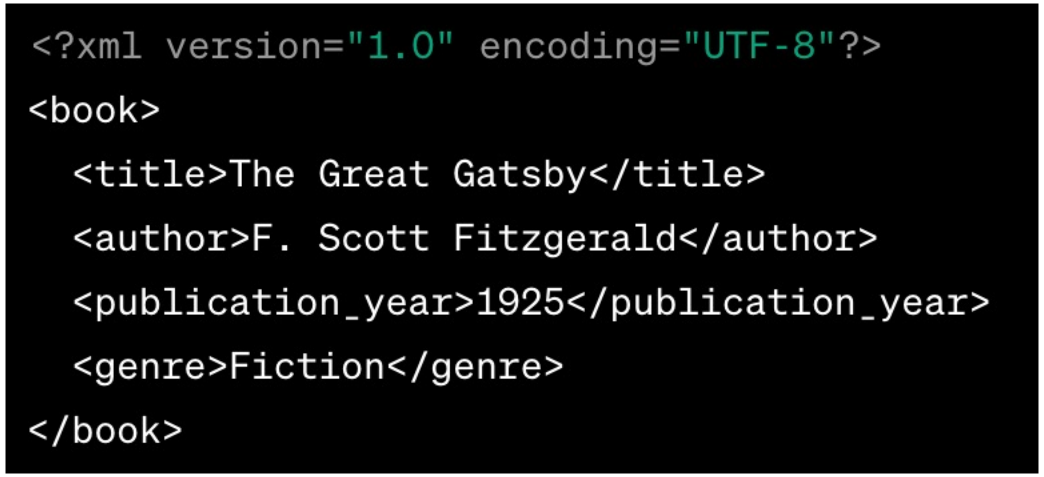
Instructions:

- Take the XML document from Exercise 1 and create an XSLT stylesheet.
- Write the XSLT code to transform the XML data into an HTML representation.
- Apply the XSLT transformation to generate an HTML file and view it in a web browser.
- Verify that the XML data is correctly transformed and displayed in the HTML format.

EXERCISE 6



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1
  <xsl:template match="/">
    <html>
      <body>
        <h1>Book Details</h1>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="book">
    <div>
      <h2><xsl:value-of select="title"/></h2>
      <p>Author: <xsl:value-of select="author"/></p>
      <p>Publication Year: <xsl:value-of select="publication_year"/></p>
      <p>Genre: <xsl:value-of select="genre"/></p>
    </div>
  </xsl:template>
</xsl:stylesheet>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <title>The Great Gatsby</title>
  <author>F. Scott Fitzgerald</author>
  <publication_year>1925</publication_year>
  <genre>Fiction</genre>
</book>
```

XML Schema Fundamentals

Best Practices

XML BEST PRACTICES

1. Well-formedness:
   - Ensure that your XML documents are well-formed, meaning they adhere to the basic syntax rules of XML. This includes properly nesting elements, using opening and closing tags correctly, and escaping special characters when necessary.
2. Validity:
   - Validate your XML documents against a schema (XSD) or Document Type Definition (DTD) to ensure they conform to a predefined structure and data rules. This helps maintain data integrity and allows for easier integration with other systems.
3. XML Namespaces:
   - Use XML namespaces to avoid naming conflicts when combining XML vocabularies or integrating with external namespaces. Assign unique namespace prefixes and declare them appropriately in your XML documents.
4. Consistent Naming Conventions:
   - Adopt a consistent naming convention for elements, attributes, and namespaces in your XML documents. Clear and meaningful names enhance readability and make it easier for others to understand and work with your XML.
5. Avoid Mixing Data and Presentation:
   - XML is primarily a data representation format. Avoid mixing presentation-related details, such as styling or formatting information, with the data itself. Use separate mechanisms, such as XSLT or CSS, for presentation purposes.

XML BEST PRACTICES

6. Use CDATA Sections for Escaping Special Characters:
   - When including text that contains special characters, such as angle brackets or ampersands, consider using CDATA (Character Data) sections to escape the characters. This ensures they are treated as raw text and not interpreted as XML markup.
7. Documentation and Comments:
   - Include documentation and comments within your XML documents to provide insights into the structure, purpose, and usage of the XML data. This helps other developers and users understand and work with the XML effectively.
8. Avoid Excessive Nesting:
   - While nesting is a fundamental concept in XML, excessive nesting can make your XML documents complex and hard to understand. Keep the nesting levels reasonable and consider using attributes or separate elements instead of excessive nesting when appropriate.
9. Compactness and Efficiency:
   - Strive for XML documents that are compact and efficient. Minimize unnecessary whitespace, use self-closing tags when applicable, and avoid redundant or repetitive data to reduce the size and improve parsing performance.
10. Encoding and Character Sets:
    - Ensure that you specify the correct character encoding (e.g., UTF-8) in the XML declaration to support the full range of characters. Consistently use the same encoding across all XML documents to prevent encoding-related issues.

XML BEST PRACTICES

11. Versioning:
    - If you anticipate changes or updates to your XML schema over time, consider incorporating versioning mechanisms to maintain backward compatibility and manage schema evolution.
12. Error Handling and Robustness:
    - Implement proper error handling mechanisms in your XML processing code to gracefully handle exceptions, detect and report validation errors, and provide meaningful error messages to users or system integrators.

XML Schema Fundamentals

Query

academy

XML QUERY

1. XPath (XML Path Language): XPath is a language used to navigate through the structure of an XML document and select nodes based on their location or specific criteria. It provides a concise syntax to traverse elements, attributes, and other parts of an XML document. XPath expressions use path expressions, filters, and functions to specify the desired nodes.
2. XQuery (XML Query Language): XQuery is a more advanced and expressive query language for XML. It allows complex querying, joining, filtering, and transformation of XML data. XQuery supports powerful constructs such as FLWOR expressions (For, Let, Where, Order by, Return), which enable iterations and conditional processing over XML nodes.
3. SQL/XML: SQL/XML is an extension to SQL (Structured Query Language) that provides XML query capabilities. It allows you to combine SQL operations with XML constructs to query XML data stored in a relational database.
4. XSLT (Extensible Stylesheet Language Transformations): While primarily a transformation language, XSLT can also be used for querying XML. By applying templates and using XPath expressions, you can select and manipulate specific portions of an XML document during the transformation process.

EXERCISE 7

Using XPath to query XML

Instructions:

- Go to: https://codebeautify.org/Xpath-Tester

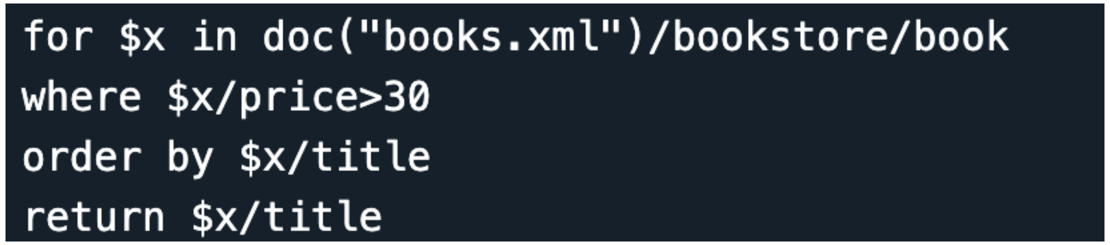
- To select all book titles:
  - XPath expression: /bookstore/book/title
- To select the author of the first book:
  - XPath expression: /bookstore/book[1]/author
- To select books published after 1950:
  - XPath expression: /bookstore/book[publication_year > 1950]

```
<bookstore>
  <book>
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
    <publication_year>1925</publication_year>
  </book>
  <book>
    <title>To Kill a Mockingbird</title>
    <author>Harper Lee</author>
    <publication_year>1960</publication_year>
  </book>
</bookstore>
```

XQUERY

Using XQuery to query XML is like SQL for DB
Same XML:

- To retrieve all books with publication years:
  - XQuery expression: for $book in /bookstore/book return $book/publication_year
- To retrieve titles of books by a specific author:
  - XQuery expression: for $book in /bookstore/book[author = "F. Scott Fitzgerald"] return $book/title
- To retrieve the number of books published in each year:
  - XQuery expression: for $year in distinct-values(/bookstore/book/publication_year) return <year count="{count(/bookstore/book[publication_year = $year])}">{$year}</year>



```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

XML Schema Fundamentals

Quiz

QUESTION 1: WHICH OF THE FOLLOWING IS NOT A VALID XML ELEMENT NAMING RULE?

a) Element names must start with a letter or underscore.
b) Element names can contain numbers and hyphens.
c) Element names are case-insensitive.
d) Element names cannot contain spaces.

academy

QUESTION 2: WHAT IS THE PURPOSE OF XML NAMESPACES?

a) To define the structure and content of an XML document.
b) To provide a unique identifier for an XML document.
c) To avoid naming conflicts in XML documents.
d) To specify the encoding of an XML document.

QUESTION 3: WHICH XML TECHNOLOGY IS USED FOR TRANSFORMING XML DATA INTO DIFFERENT FORMATS, SUCH AS HTML OR PDF?

a) XSLT (Extensible Stylesheet Language Transformations)
b) XPath (XML Path Language)
c) XQuery (XML Query Language)
d) XSD (XML Schema Definition)

academy

QUESTION 4: WHICH XML PARSING MODEL CREATES A TREE-LIKE REPRESENTATION OF THE ENTIRE XML DOCUMENT IN MEMORY?

a) DOM (Document Object Model)
b) SAX (Simple API for XML)
c) StAX (Streaming API for XML)
d) XSLT (Extensible Stylesheet Language Transformations)

QUESTION 5: WHAT DOES DTD STAND FOR IN XML?

a) Document Transformation Definition
b) Document Type Description
c) Data Type Definition
d) Document Type Definition

XML Schema Fundamentals

Namespaces

academy

Capgemini

XML NAMESPACES

- XML namespaces are used to differentiate elements and attributes that have the same name but belong to different vocabularies or schema definitions.
- By associating elements with namespaces, XML documents can incorporate elements from different sources or define their own custom elements without conflicting with other names.

- A namespace is declared using the **xmlns** attribute, which stands for "XML Namespace."
  - In root element
- Default Namespace

```
<root xmlns="http://www.example.com/mynamespace">
  <element1>Value 1</element1>
  <element2>Value 2</element2>
</root>
```

- Namespace Scope:
  - Namespaces have a scope within an XML document, defined by the elements where they are declared. If a namespace is declared in a parent element, it applies to all descendant elements within that parent, unless overridden by another namespace declaration.

XML NAMESPACES

- Qualified Names: To associate elements and attributes with namespaces, qualified names are used.
  - A qualified name consists of a namespace prefix and a local name, separated by a colon.
  - The namespace prefix is defined in the namespace declaration and maps to a specific namespace URI.

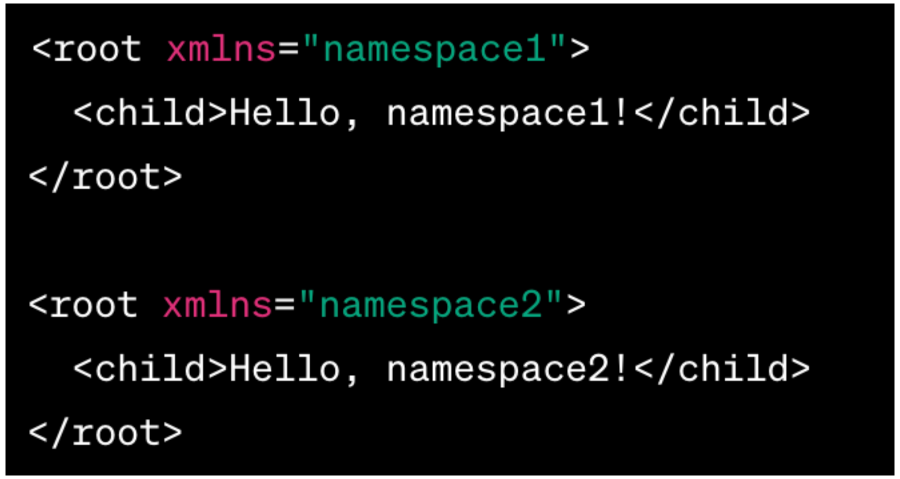
```
<root xmlns:ns="http://www.example.com/mynamespace">
  <ns:element1>Value 1</ns:element1>
  <ns:element2>Value 2</ns:element2>
</root>
```

  - Namespace Prefixes:
    - not required to be specific keywords like "ns" or "xmlns".
    - valid, as long as they are declared and associated with the appropriate namespace URI.

```
<root xmlns:custom="http://www.example.com/mynamespace">
  <custom:element1>Value 1</custom:element1>
  <custom:element2>Value 2</custom:element2>
</root>
```

XML NAMESPACES : CHAMELEON EFFECT

- Behavior in XML where the interpretation of element names can change depending on the context of the namespace declarations in an XML document.
  - This effect occurs when an XML document uses the same element names but associates them with different namespaces.



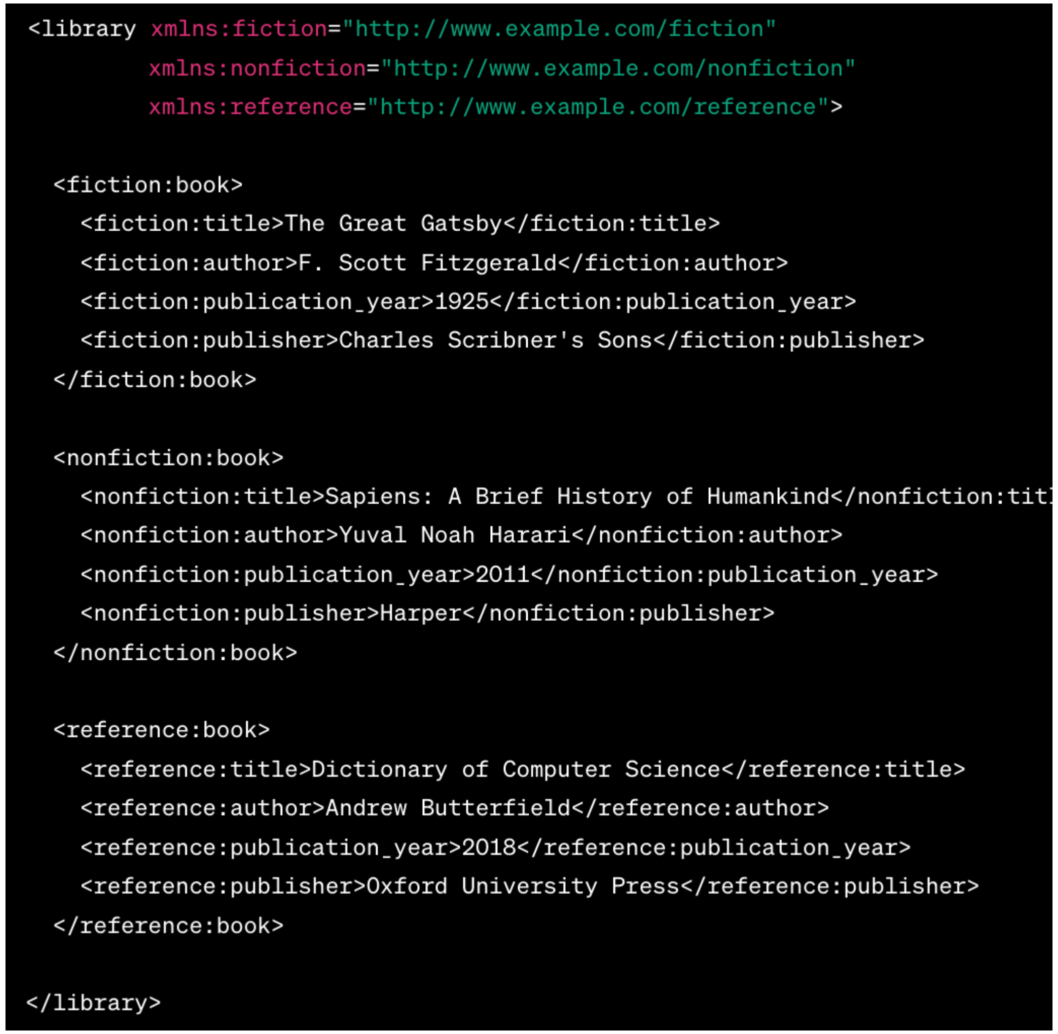
```
<root xmlns="namespace1">
  <child>Hello, namespace1!</child>
</root>

<root xmlns="namespace2">
  <child>Hello, namespace2!</child>
</root>
```

EXERCISE 8

Create namespaces for your book collection

Instructions:

- Create an XML document representing a library catalog (use from earlier exercises).
- Use namespaces to distinguish between different types of books, such as fiction, non-fiction, and reference books.
- Each book should have elements like "title," "author," "publication_year," and "publisher."
- Use different namespace prefixes for each book category.



```
<library xmlns:fiction="http://www.example.com/fiction"
         xmlns:nonfiction="http://www.example.com/nonfiction"
         xmlns:reference="http://www.example.com/reference">

  <fiction:book>
    <fiction:title>The Great Gatsby</fiction:title>
    <fiction:author>F. Scott Fitzgerald</fiction:author>
    <fiction:publication_year>1925</fiction:publication_year>
    <fiction:publisher>Charles Scribner's Sons</fiction:publisher>
  </fiction:book>

  <nonfiction:book>
    <nonfiction:title>Sapiens: A Brief History of Humankind</nonfiction:tit
    <nonfiction:author>Yuval Noah Harari</nonfiction:author>
    <nonfiction:publication_year>2011</nonfiction:publication_year>
    <nonfiction:publisher>Harper</nonfiction:publisher>
  </nonfiction:book>

  <reference:book>
    <reference:title>Dictionary of Computer Science</reference:title>
    <reference:author>Andrew Butterfield</reference:author>
    <reference:publication_year>2018</reference:publication_year>
    <reference:publisher>Oxford University Press</reference:publisher>
  </reference:book>

</library>
```

XML Schema Fundamentals

Inheritance

academy

XML INHERITANCE

- Superclass and Subclass:
  - Inheritance involves two key classes: a superclass (also known as a base class or parent class) and one or more subclasses (also known as derived classes or child classes). The superclass serves as a template or blueprint that defines common attributes and behaviors, while subclasses extend or specialize the superclass by adding additional attributes or behaviors specific to their needs.
- Inheriting Properties and Behaviors:
  - A subclass inherits all the properties (attributes and methods) defined in its superclass. This means that the subclass can access and use those properties without redefining them. Inheritance allows subclasses to reuse existing code and avoid redundancy by inheriting common functionality from the superclass.
- Extending and Overriding:
  - Subclasses have the ability to extend the functionality inherited from the superclass by adding new properties or methods. They can define additional attributes and behaviors that are specific to the subclass. Subclasses can also override (modify) inherited methods from the superclass to provide different implementations that are more appropriate for their specific context.
- Inheritance Hierarchy:
  - Inheritance can be organized into a hierarchy or tree-like structure, with the superclass at the top and subclasses branching out from it. This hierarchy allows for the classification and organization of classes based on their shared characteristics and relationships. Subclasses can further serve as superclasses for other subclasses, creating a chain of inheritance.

XML INHERITANCE

- Code Reusability:
  - Inheritance promotes code reusability, as common properties and behaviors defined in the superclass are automatically available to all subclasses. This reduces code duplication and makes maintenance easier, as changes made to the superclass automatically propagate to all subclasses.
- Polymorphism:
  - Inheritance is closely related to the concept of polymorphism. Polymorphism allows objects of different classes, but related through inheritance, to be treated as instances of their superclass. This allows for more generic and flexible programming, as the same code can operate on objects of different subclasses without needing to know their specific types.

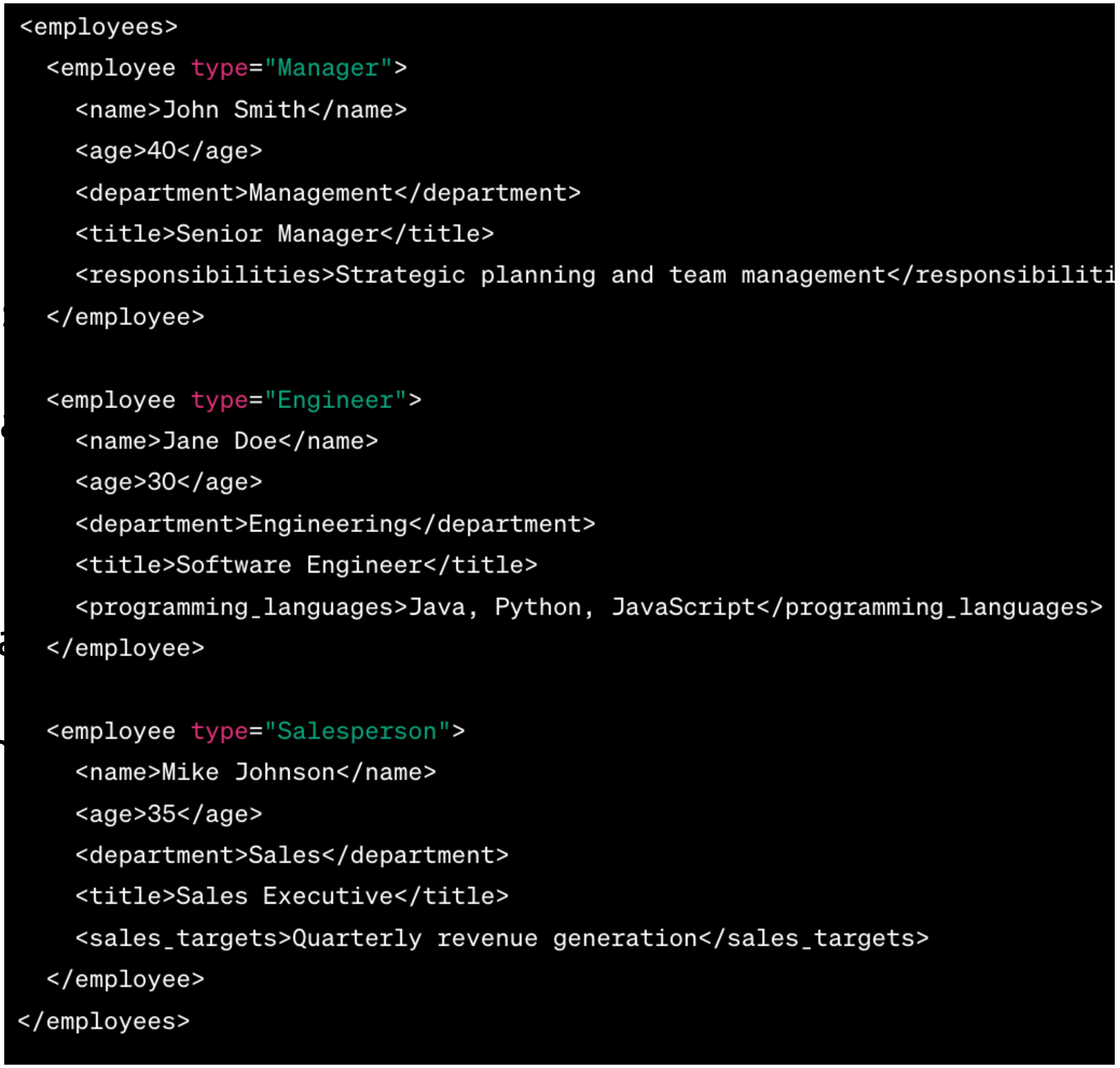
- Inheritance is a powerful concept that facilitates code organization, code reuse, and extensibility in object-oriented programming. By defining a hierarchical relationship between classes and allowing subclasses to inherit and extend the properties and behaviors of their superclasses, inheritance helps create more structured, modular, and maintainable code.

EXERCISE 9

Create Inheritance

Instructions:

- Create an XML document that represents a company'
employee records.
- Each employee should have common attributes like n
age, and department.
- Use inheritance to define specialized employee types
as "Manager," "Engineer," and "Salesperson."
- Each specialized employee type should have additiona
attributes specific to their role.
- Create multiple instances of each employee type with
the XML document.



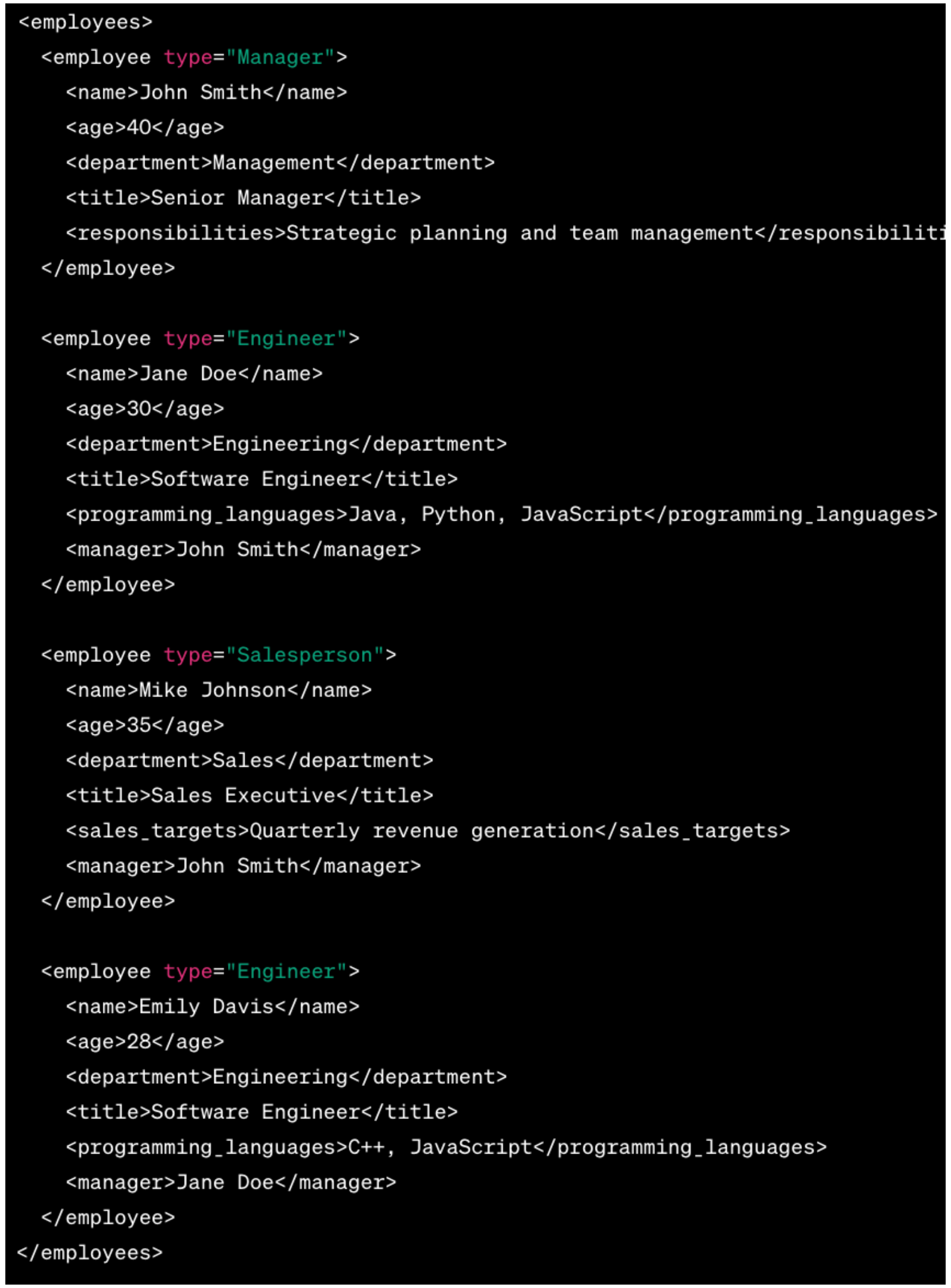
```xml
<employees>
  <employee type="Manager">
    <name>John Smith</name>
    <age>40</age>
    <department>Management</department>
    <title>Senior Manager</title>
    <responsibilities>Strategic planning and team management</responsibiliti
  </employee>

  <employee type="Engineer">
    <name>Jane Doe</name>
    <age>30</age>
    <department>Engineering</department>
    <title>Software Engineer</title>
    <programming_languages>Java, Python, JavaScript</programming_languages>
  </employee>

  <employee type="Salesperson">
    <name>Mike Johnson</name>
    <age>35</age>
    <department>Sales</department>
    <title>Sales Executive</title>
    <sales_targets>Quarterly revenue generation</sales_targets>
  </employee>
</employees>
```

EXERCISE 10

Create Inheritance

Instructions:

- Extend the XML document from Exercise 1 to include a hierarchy of employee roles.
- For example, a "Manager" can have subordinates who are "Engineers" or "Salespeople."
- Use inheritance to represent this hierarchy within the XML document.
- Each employee should have a reference to their manager, if applicable.



```xml
<employees>
  <employee type="Manager">
    <name>John Smith</name>
    <age>40</age>
    <department>Management</department>
    <title>Senior Manager</title>
    <responsibilities>Strategic planning and team management</responsibiliti
  </employee>

  <employee type="Engineer">
    <name>Jane Doe</name>
    <age>30</age>
    <department>Engineering</department>
    <title>Software Engineer</title>
    <programming_languages>Java, Python, JavaScript</programming_languages>
    <manager>John Smith</manager>
  </employee>

  <employee type="Salesperson">
    <name>Mike Johnson</name>
    <age>35</age>
    <department>Sales</department>
    <title>Sales Executive</title>
    <sales_targets>Quarterly revenue generation</sales_targets>
    <manager>John Smith</manager>
  </employee>

  <employee type="Engineer">
    <name>Emily Davis</name>
    <age>28</age>
    <department>Engineering</department>
    <title>Software Engineer</title>
    <programming_languages>C++, JavaScript</programming_languages>
    <manager>Jane Doe</manager>
  </employee>
</employees>
```

XML Schema Fundamentals

Security

academy

XML SECURITY

1. Avoid Exposing Sensitive Information:
   - Do not include sensitive information such as passwords, personally identifiable information (PII), or sensitive business data in XML documents unless necessary. Instead, securely store such information separately and transmit it using secure channels when required.
2. Avoid XML Entity Expansion:
   - XML entity expansion allows for the inclusion of external entities within an XML document, which can lead to security vulnerabilities such as XXE (XML External Entity) attacks. Disable or restrict entity expansion to prevent these attacks.
3. Prevent XML Injection:
   - Avoid incorporating untrusted data directly into XML structures without proper validation and encoding. Validate and sanitize all user-supplied input to prevent XML injection attacks, which can manipulate the XML document's structure or execute malicious code.
4. Limit XML Parsing Capabilities:
   - XML parsers often provide various features and options that can introduce security risks. Disable unnecessary features, such as entity expansion, document type definitions (DTD), external entity resolution, and external style sheet processing, unless explicitly required.
5. Validate XML Structure:
   - Use XML Schema Definitions (XSD) or Document Type Definitions (DTD) to validate the structure and integrity of incoming XML documents. This helps prevent malformed or malicious XML data from being processed.

XML SECURITY

6. Secure XML Processing Libraries:
    - Ensure that the XML processing libraries or frameworks used in your applications are up to date and properly patched to address any known security vulnerabilities. Keep track of security advisories and updates released by the library providers.
7. Secure Configuration and Access Control:
    - Follow secure configuration practices for XML processing components and servers. Configure access controls to limit the privileges and permissions of XML processors and ensure proper authentication and authorization mechanisms are in place.
8. Implement XML Encryption and Digital Signatures:
    - When transmitting sensitive XML data or exchanging XML documents, consider using XML encryption and digital signatures to ensure confidentiality, integrity, and authenticity of the data.
9. Use Secure Communication Channels:
    - Transmit XML data over secure communication channels, such as using HTTPS (HTTP over SSL/TLS) protocols, to protect data during transit.
10. Regularly Update and Patch Systems:
    - Stay updated with the latest security patches, bug fixes, and updates for your XML processing software, libraries, and frameworks. Regularly review and update your security practices as new vulnerabilities are discovered.

Capgemini

XML Schema Fundamentals

Data types exercises

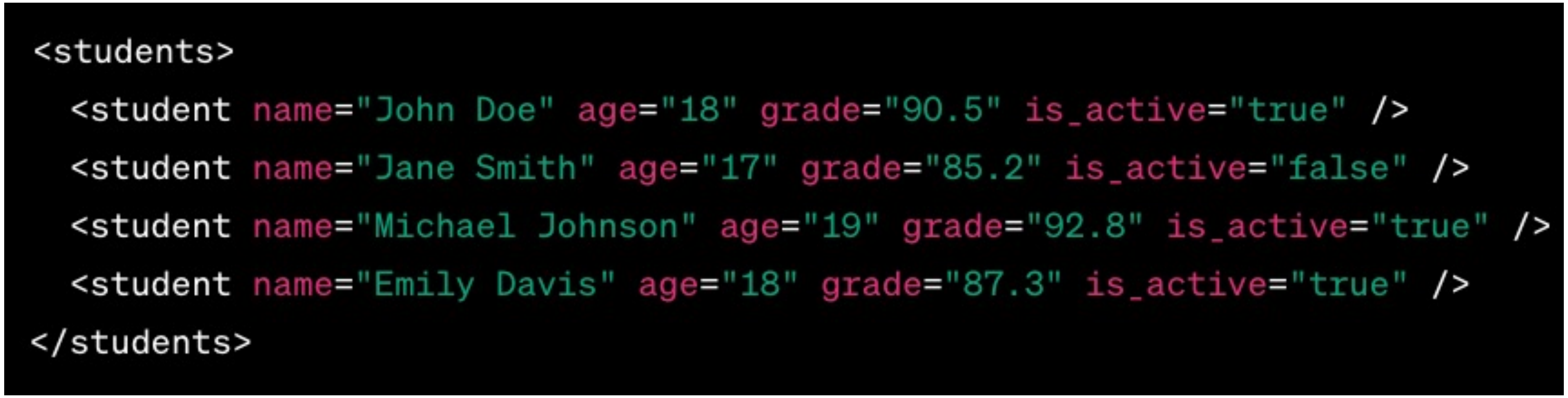
academy

EXERCISE 11

Create datatypes

Instructions:

- Create an XML document that represents a student record.
- Each student should have attributes like "name" (string), "age" (integer), "grade" (float), and "is_active" (boolean).
- Add multiple instances of students with different values for each attribute.



```
<students>
  <student name="John Doe" age="18" grade="90.5" is_active="true" />
  <student name="Jane Smith" age="17" grade="85.2" is_active="false" />
  <student name="Michael Johnson" age="19" grade="92.8" is_active="true" />
  <student name="Emily Davis" age="18" grade="87.3" is_active="true" />
</students>
```

About Capgemini Academy

Capgemini is a global leader in partnering with companies to transform and manage their business by harnessing the power of technology. The Group is guided everyday by its purpose of unleashing human energy through technology for an inclusive and sustainable future. It is a responsible and diverse organization of 270,000 team members in nearly 50 countries. With its strong 50 year heritage and deep industry expertise, Capgemini is trusted by its clients to address the entire breadth of their business needs, from strategy and design to operations, fuelled by the fast evolving and innovative world of cloud, data, AI, connectivity, software, digital engineering and platforms. The Group reported in 2020 global revenues of €16 billion.

Get the Future You Want | academy.capgemini.nl