

## Ficha Prática 5

### Exe 1

a) Crie uma função XQuery que receba como parâmetro o nome de um autor e que retorne todos os livros desse autor.

The screenshot displays the XML Editor (oXygen) interface. The main editor shows an XML document named 'bookstore.xml' with the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J. K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

The 'XPath/XQuery Builder' panel on the right shows the following XQuery function:

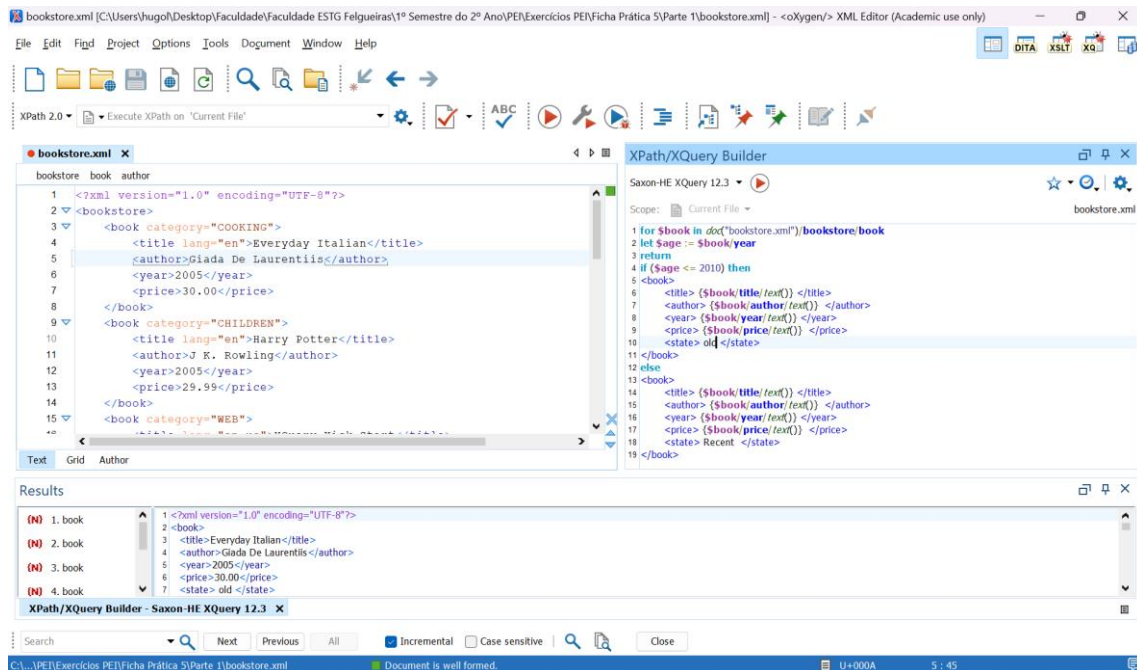
```
1 declare function local:books-by-author($name as element)*
2 {
3   for $livro in doc("bookstore.xml")/book
4   where some $ba in $livro/author satisfies ($ba=$name)
5   return $livro/title
6 };
7 let $n := local:books-by-author("J. K. Rowling")
8 return $n
```

The 'Results' panel at the bottom shows the output of the query, which is a list of titles:

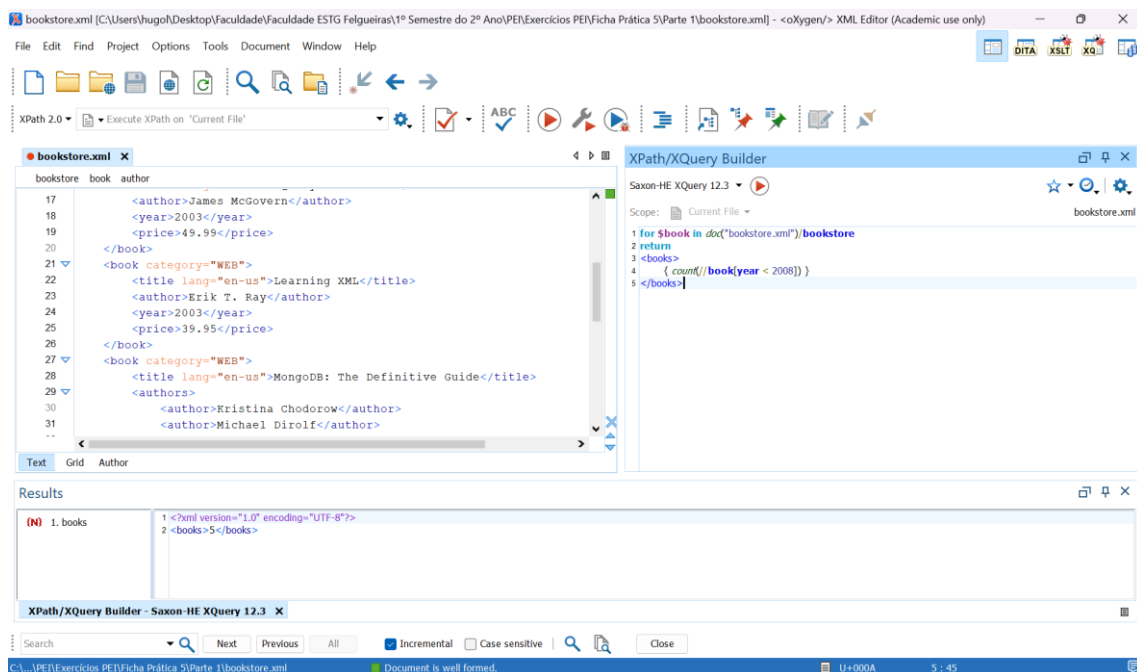
```
1. title
2. <title lang="en">Harry Potter</title>
```

The status bar at the bottom indicates that the document is well formed.

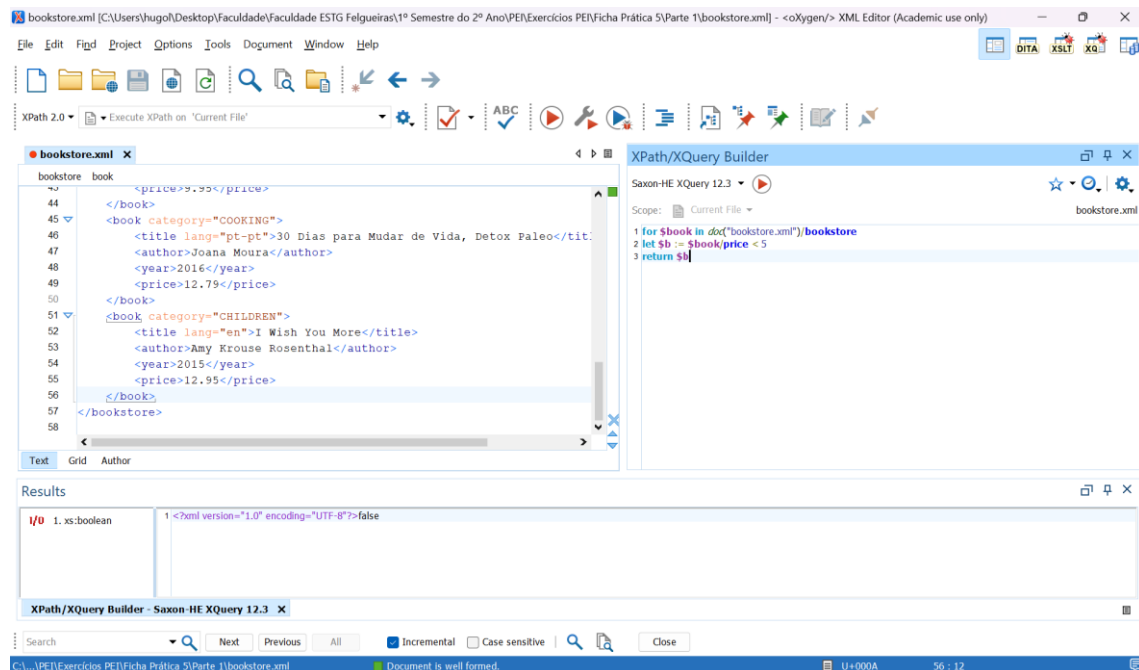
b) Crie uma função XQuery que retorne todos os livros e que caracterize o estado de cada um deles em “antigo” ou “recente”. O estado “antigo” deverá corresponder aos livros publicados até 2010 (inclusive) e o estado “recente” aos livros posteriores a 2010. Para isso, inclua um elemento “state” em cada elemento “book”.



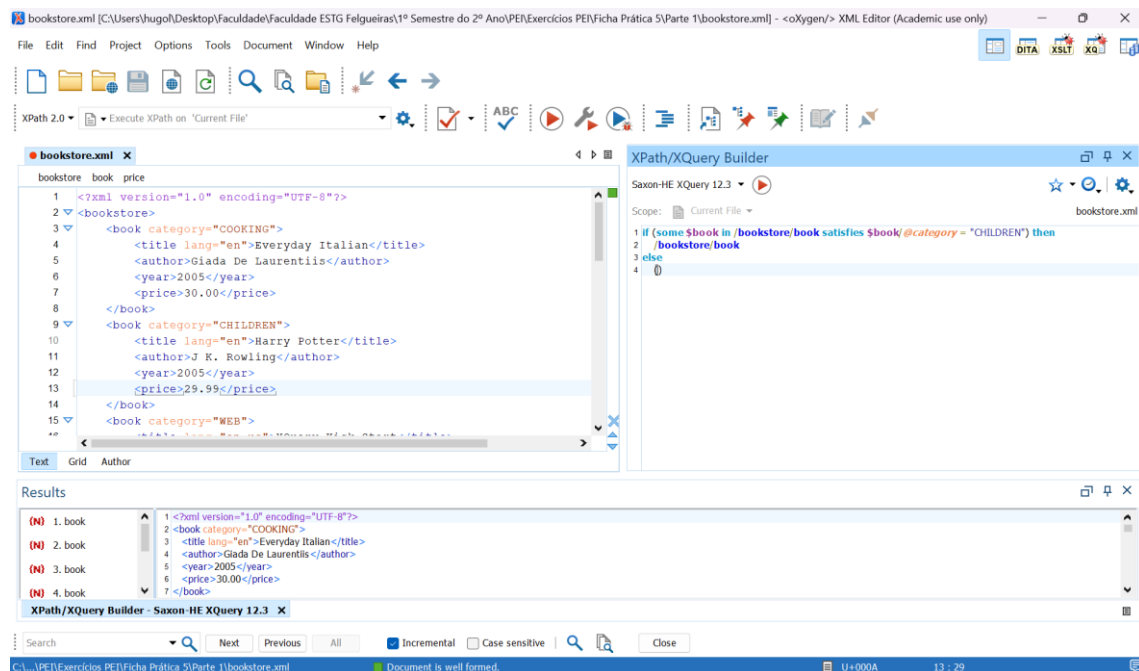
c) Escreva uma expressão XQuery que permita determinar o número de livros que são anteriores a 2008.



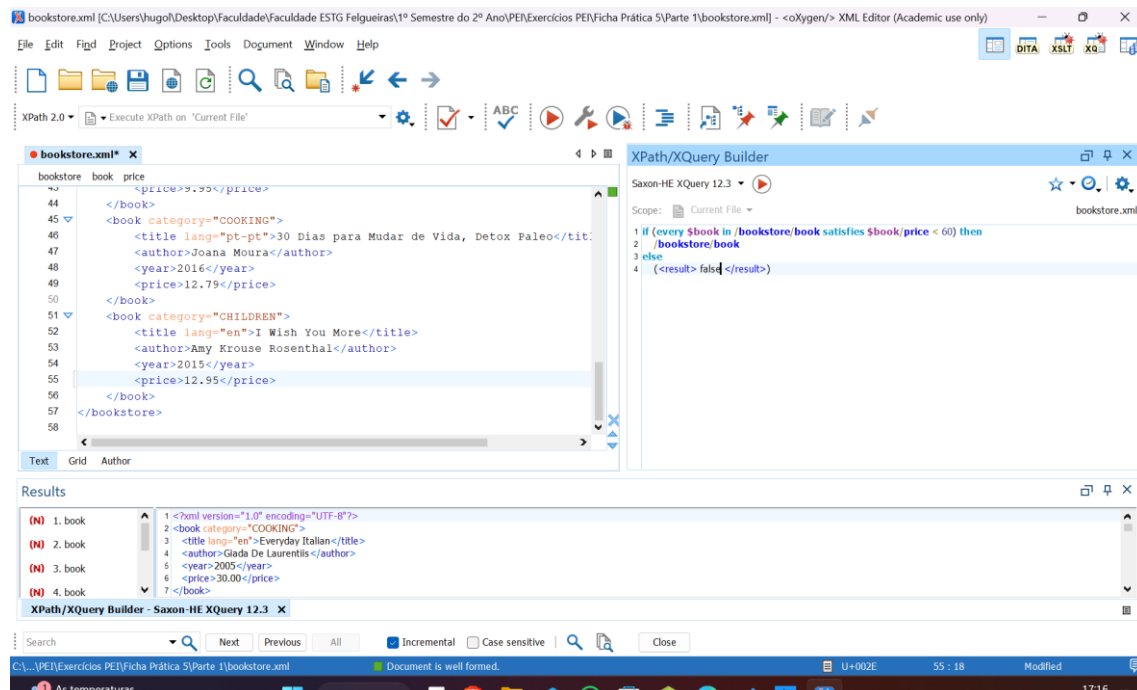
d) Escreva uma expressão quantificada que permita determinar se existem livros que custam menos do que 5.



e) Escreva uma expressão XQuery que verifique se algum dos livros é da categoria "CHILDREN" e, em caso afirmativo, que retorne todos os livros da *bookstore*.

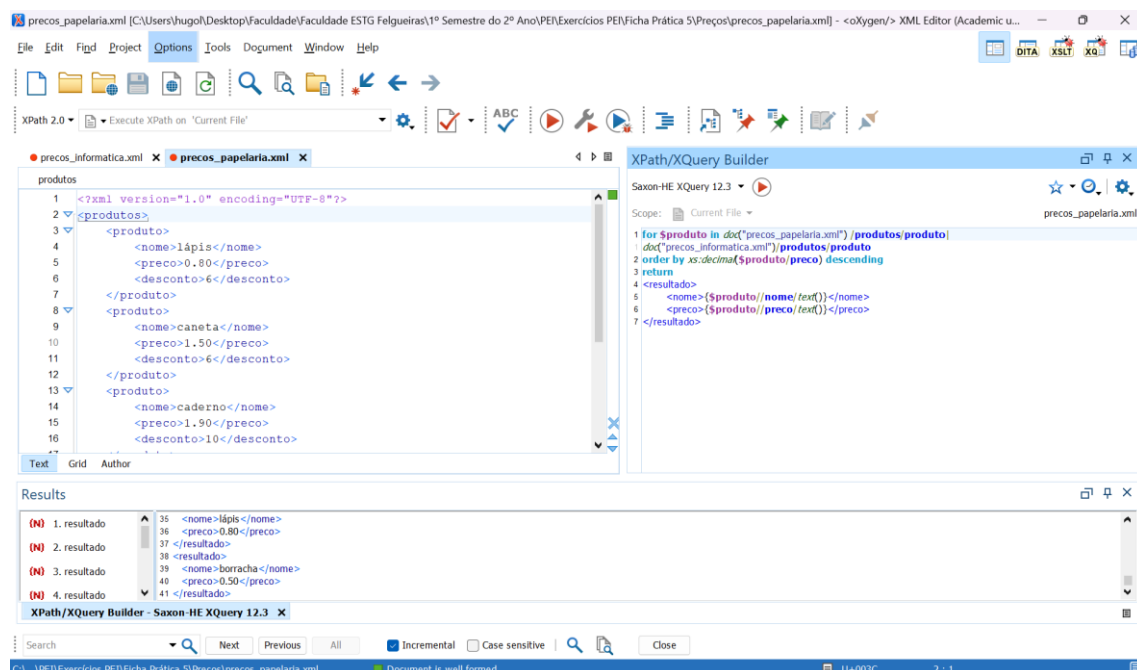


f) Escreva uma expressão XQuery que verifique se todos os livros custam menos do que 60 e, em caso afirmativo, liste-os.



2. Crie uma diretoria Preços e coloque os ficheiros `precos_papelaria.xml` e `precos_informatica.xml` nessa diretoria. Construa expressões XQuery que permitam (dica: considere a função `collection`):

a) Listar o nome dos produtos por ordem decrescente de preço



b) Indicar o preço médio dos produtos.

The screenshot shows the XML Editor with two files: `precos_informatica.xml` and `precos_papelaria.xml`. The `precos_informatica.xml` file is open, displaying an XML structure with a root element `produtos` containing three `produto` elements. The first `produto` element has attributes `nome` and `preco`, and a `desconto` element. The second `produto` element has attributes `nome` and `preco`, and a `desconto` element. The third `produto` element has attributes `nome` and `preco`, and a `desconto` element.

The XPath/XQuery Builder window shows the following query:

```
1 let $produto := (doc("precos_papelaria.xml")//produto | doc("precos_informatica.xml")//produto)
2 let $avg := $produto/preco
3 return
4 <resultado>
5   <precoMedio>{avg($avg)}</precoMedio>
6 </resultado>
```

The Results window shows the output of the query:

```
1. resultado
1 <?xml version="1.0" encoding="UTF-8"?>
2 <resultado>
3   <precoMedio>29.07</precoMedio>
4 </resultado>
```

c) Indicar o valor total a pagar por uma compra de 1 lápis e 2 borrachas

The screenshot shows the XML Editor with two files: `precos_informatica.xml` and `precos_papelaria.xml`. The `precos_papelaria.xml` file is open, displaying an XML structure with a root element `produtos` containing three `produto` elements. The first `produto` element has attributes `nome` and `preco`, and a `desconto` element. The second `produto` element has attributes `nome` and `preco`, and a `desconto` element. The third `produto` element has attributes `nome` and `preco`, and a `desconto` element.

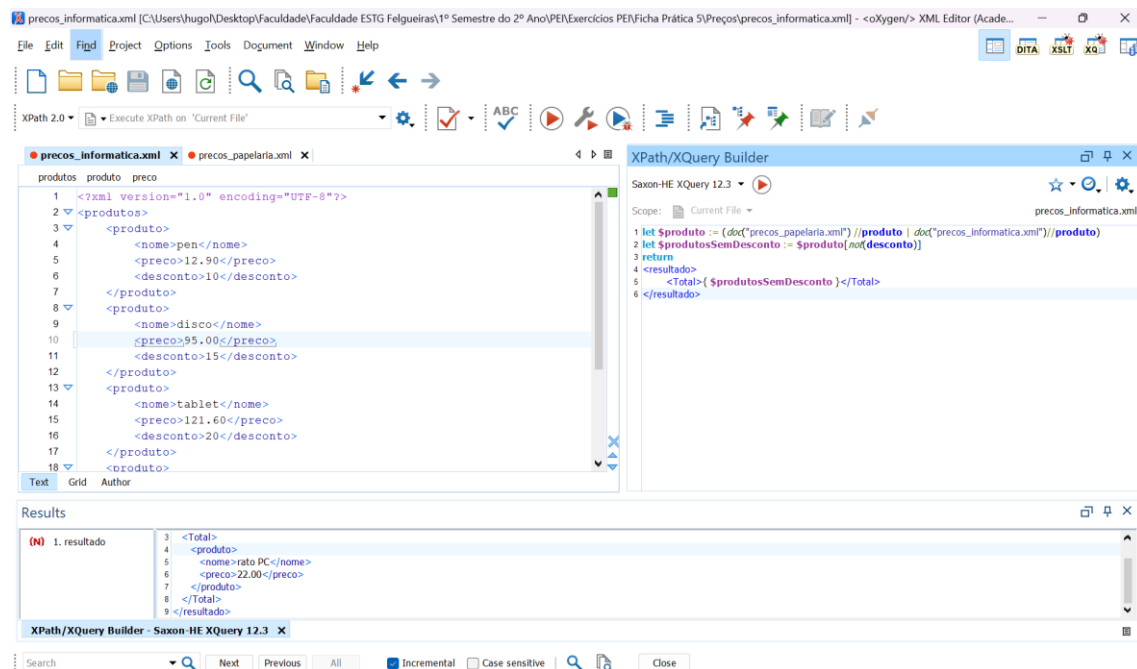
The XPath/XQuery Builder window shows the following query:

```
1 let $produto := (doc("precos_papelaria.xml")//produto | doc("precos_informatica.xml")//produto)
2 let $lapis := 0.80
3 let $borracha := 0.50
4 return
5 <resultado>
6   <Total>{($lapis + $borracha * 2)</Total>}
7 </resultado>
```

The Results window shows the output of the query:

```
1. resultado
1 <?xml version="1.0" encoding="UTF-8"?>
2 <resultado>
3   <Total>1.8</Total>
4 </resultado>
```

d) Listar o nome dos produtos que não têm desconto.



The screenshot shows the XML Editor interface with the file `precos_informatica.xml` open. The XML document structure is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<produtos>
  <produto>
    <nome>pen</nome>
    <preco>12.90</preco>
    <desconto>10</desconto>
  </produto>
  <produto>
    <nome>disco</nome>
    <preco>95.00</preco>
    <desconto>15</desconto>
  </produto>
  <produto>
    <nome>tablet</nome>
    <preco>121.60</preco>
    <desconto>20</desconto>
  </produto>
</produtos>
```

The XPath/XQuery Builder shows the following XQuery:

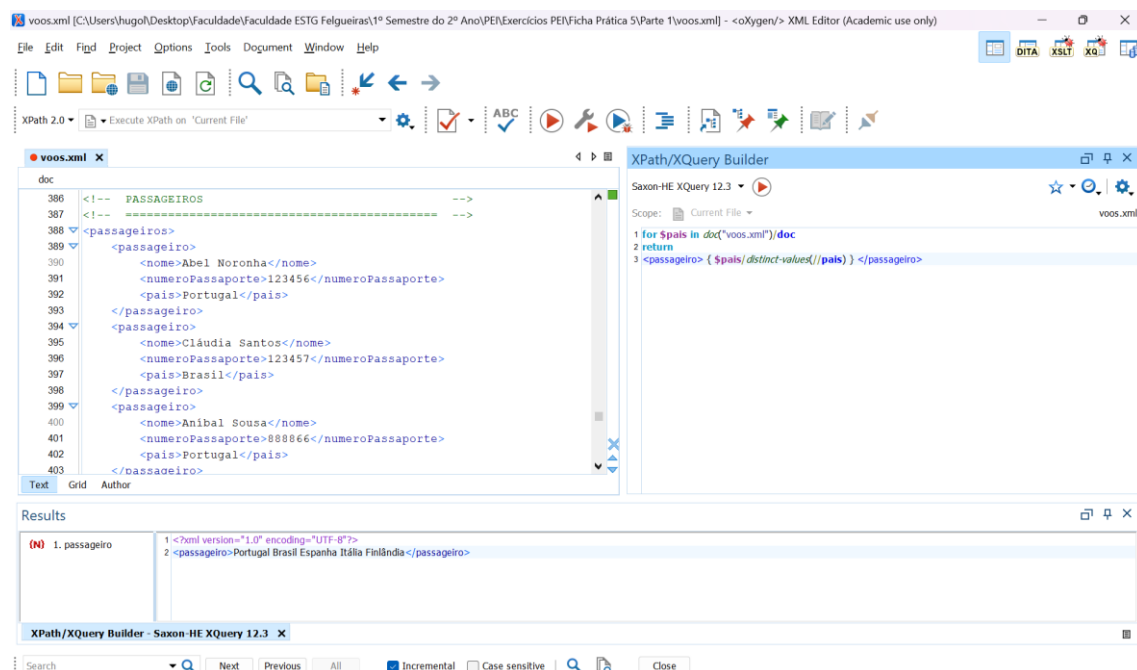
```
1 let $produto := (doc("precos_papelaria.xml")//produto | doc("precos_informatica.xml")//produto)
2 let $produtosSemDesconto := $produto[not(desconto)]
3 return
4 <resultado>
5   <Total> { $produtosSemDesconto } </Total>
6 </resultado>
```

The Results pane displays the output of the query:

```
(N) 1. resultado
3   <Total>
4     <produto>
5       <nome>rato PC</nome>
6       <preco>22.00</preco>
7     </produto>
8   </Total>
9 </resultado>
```

3. Considere o documento voos.xml fornecido na plataforma moodle. Escreva expressões XQuery que permitam realizar as seguintes consultas:

a) Listar todos os diferentes países de onde são oriundos os passageiros



The screenshot shows the XML Editor interface with the file `voos.xml` open. The XML document structure is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <!-- PASSAGEIROS -->
  <!-- -->
  <passageiros>
    <passageiro>
      <nome>Abel Noronha</nome>
      <numeroPassaporte>123456</numeroPassaporte>
      <pais>Portugal</pais>
    </passageiro>
    <passageiro>
      <nome>Cláudia Santos</nome>
      <numeroPassaporte>123457</numeroPassaporte>
      <pais>Brasil</pais>
    </passageiro>
    <passageiro>
      <nome>Anibal Sousa</nome>
      <numeroPassaporte>888866</numeroPassaporte>
      <pais>Portugal</pais>
    </passageiro>
  </passageiros>
</doc>
```

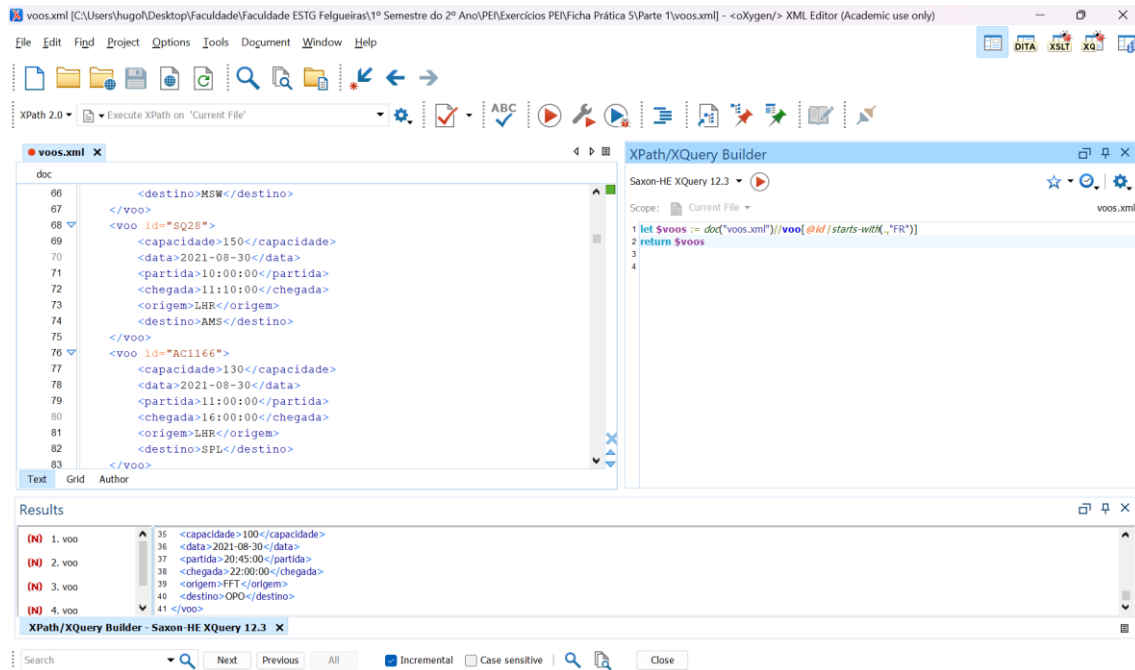
The XPath/XQuery Builder shows the following XQuery:

```
1 for $pais in doc("voos.xml")/doc
2 return
3 <passageiro> { $pais/distinct-values(//pais) } </passageiro>
```

The Results pane displays the output of the query:

```
(N) 1. passageiro
1 <?xml version="1.0" encoding="UTF-8"?>
2 <passageiro>Portugal Brasil Espanha Itália Finlândia</passageiro>
```

b) Listar todos os voos que contêm "FR" no seu id do voo.

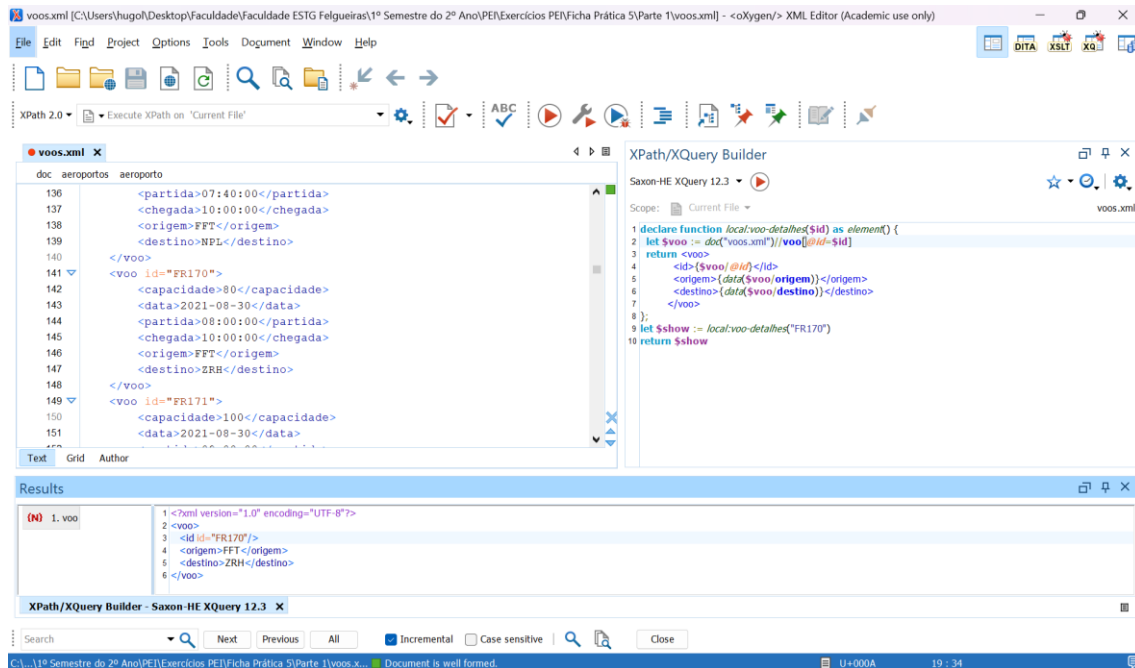


The screenshot shows the XML Editor interface with the file 'voos.xml' open. The XML content is displayed in the left pane, showing a list of flight records. The XPath/XQuery Builder pane on the right contains the following query:

```
1 let $voos := doc('voos.xml')/voo[@id starts-with('FR')]
2 return $voos
```

The Results pane at the bottom shows the output of the query, listing four flight records (1. voo, 2. voo, 3. voo, 4. voo) with their respective details.

c) Listar todos os voos que contêm "FR" no seu id do voo e que devolva o resultado com a seguinte estrutura:



The screenshot shows the XML Editor interface with the file 'voos.xml' open. The XML content is displayed in the left pane, showing a list of flight records. The XPath/XQuery Builder pane on the right contains the following query:

```
1 declare function local:voo-detalhes($id as element) {
2   let $voo := doc('voos.xml')/voo[@id=$id]
3   return <voo>
4     <id>{$voo/@id}</id>
5     <origem>{$voo/origem}</origem>
6     <destino>{$voo/destino}</destino>
7   </voo>
8 };
9 let $show := local:voo-detalhes('FR170')
10 return $show
```

The Results pane at the bottom shows the output of the query, displaying a single flight record (1. voo) with its details in a structured format.

d) Verificar se algum dos voos realizados no dia 2021-08-30 tinha como destino o aeroporto “NPL”.

The screenshot shows the XML Editor interface with the 'XPath/XQuery Builder' window open. The XML document 'voos.xml' contains flight information. The query in the builder is as follows:

```

1 for $voo in doc("voos.xml")/voo
2 where $voo/data = "2021-08-30"
3 and $voo/destino = "NPL"
4 return
5 <voo>
6 <data> { data($voo/data) } </data>
7 <destino> { data($voo/destino) } </destino>
8 </voo>

```

The 'Results' window displays the output of the query, showing four flight records (1. voo to 4. voo) that match the criteria. The status bar at the bottom indicates 'Document is well formed'.

e) Verificar se algum passageiro de nome “Santa Claus” viajou num voo nas condições indicadas na alínea anterior.

The screenshot shows the XML Editor interface with the 'XPath/XQuery Builder' window open. The XML document 'voos.xml' contains flight reservation information. The query in the builder is as follows:

```

1 for $voo in doc("voos.xml")/voo
2 for $passageiro in doc("voos.xml")/passageiro
3 where $voo/data = "2021-08-30"
4 and $voo/destino = "NPL"
5 and $passageiro/nome = "Santa Claus"
6 return
7 <voo>
8 <data> { data($voo/data) } </data>
9 <destino> { data($voo/destino) } </destino>
10 <nome> { data($passageiro/nome) } </nome>
11 </voo>

```

The 'Results' window displays the output of the query, showing four flight records (1. voo to 4. voo) that match the criteria. The status bar at the bottom indicates 'Document is well formed'.



f) Listar todos os voos realizados no dia 2021-08-30 com origem no aeroporto “NPL”

The screenshot shows the XML Editor interface with the file 'voos.xml' open. The XPath/XQuery Builder panel on the right contains the following query:

```

1 for $voo in doc("voos.xml")//voo
2 for $passageiro in doc("voos.xml")//passageiro
3 where $voo/data = "2021-08-30"
4 and $voo/origem = "NPL"
5 return
6 <voo>
7 <data> {data($voo/data)} </data>
8 <origem> {data($voo/origem)} </origem>
9 </voo>

```

The Results panel shows four results, each representing a flight record with its date and origin.

Result	XML Snippet
1. voo	<data>2021-08-30</data> <origem>ZRH</origem>
2. voo	<data>2021-08-30</data> <origem>ZRH</origem>
3. voo	<data>2021-08-30</data> <origem>ZRH</origem>
4. voo	<data>2021-08-30</data> <origem>ZRH</origem>

g) Contar quantos voos cumprem as condições indicadas na alínea anterior

The screenshot shows the XML Editor interface with the file 'voos.xml' open. The XPath/XQuery Builder panel on the right contains the following query:

```

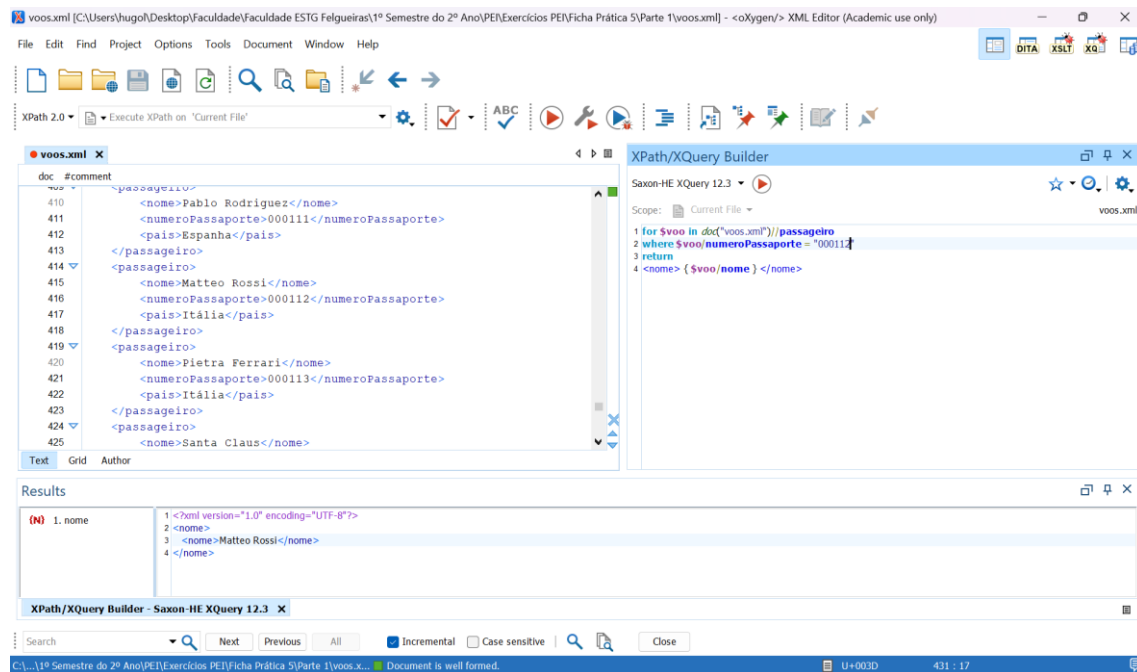
1 let $voo := doc("voos.xml")//voo
2 where $voo/data = "2021-08-30"
3 and $voo/origem = "NPL"
4 return
5 <voo>
6 <contagem> {count($voo/data and $voo/origem)} </contagem>
7 </voo>

```

The Results panel shows one result, which is a single XML element representing the count of flights.

Result	XML Snippet
1. voo	<contagem>1</contagem>

h) Listar o nome do passageiro que está associado ao passaporte com o número “000112”.



i) Listar o nome do passageiro que está associado à reserva com id igual a 1

j) Construir uma função que recebe como parâmetro o número de reserva e que devolva o nome do passageiro que lhe está associado.

k) Construir uma função que deduza 10% ao valor da taxa de um dado aeroporto (considere a abreviatura como representativa do aeroporto).

l) Aplicar um desconto de 10% ao valor da taxa de todos os aeroportos e retorne o novo valor da taxa para cada um dos aeroportos. Utilize a função definida anteriormente.