```
(::::::::::::::::::::::::::::::)
(:     q01.xq                 :)


doc("bib.xml")/bib/book/year

(:                            :)
(::::::::::::::::::::::::::::::)
(:     q02.xq                 :)

doc("bib.xml")/bib//first


(:                            :)
(::::::::::::::::::::::::::::::)
(:     q03.xq                 :)


doc("bib.xml")/bib/book/@price

(:                            :)
(::::::::::::::::::::::::::::::)
(:     q04.xq                 :)


doc("bib.xml")//author


(:                            :)
(::::::::::::::::::::::::::::::)
(:     q05.xq                 :)


doc("bib.xml")/bib/book/author/text()
(:                            :)
(::::::::::::::::::::::::::::::)
(:     q06.xq                 :)


doc("bib.xml")/bib/book/author[first]


(:                            :)
(::::::::::::::::::::::::::::::)
(:     q07.xq                 :)

doc("bib.xml")/bib/book/author/../author


(:                            :)
(::::::::::::::::::::::::::::::)
(:     q08.xq                 :)


doc("bib.xml")/bib/book/author[./first/../last]


(:                            :)
(::::::::::::::::::::::::::::::)
(:     q09.xq                 :)

doc("bib.xml")/bib/book/author[first][last]
```

```
(:                            :)
(::::::::::::::::::::::::::::::)
(:     q11.xq                 :)
(:     basic FLWR expression  :)

for $x in doc("bib.xml")/bib/book
where $x/year/text() > 1995
return $x/title



(: try also:
for $x in doc("bib.xml")/bib/book
where $x/@year > 1995
return $x/title
:)


(: Same, more geek'ish
for $x in doc("bib.xml")/bib/book[year/text() > 1995] /t
itle
return $x
:)

(: Even better:
doc("bib.xml")/bib/book[year/text() > 1995] /title
:)


(:                            :)
(::::::::::::::::::::::::::::::)
(:     q12.xq                 :)
(:     return clause          :)

for $x in doc("bib.xml")/bib/book
return <answer>
          <title> { $x/title/text() } </title>
          <year>{ $x/year/text() } </year>
       </answer>


(: why do we use { and } ?  what happens without them ?
:)


(:                            :)
(::::::::::::::::::::::::::::::)
(:     q13.xq                 :)
(:     Nesting                :)

for $b in doc("bib.xml")/bib,
    $a in $b/book[year/text()=1995]/author
return <result>
         { $a,
           for $t in $b/book[author/text()=$a/text()]/t
itle
           return $t
         }
       </result>


(:                            :)
(::::::::::::::::::::::::::::::)
(:     q14.xq                 :)
```

```
(:     aggregates             :)

for $x in doc("bib.xml")/bib/book
where count($x/author)>2
return $x


(:
Aggregates in XQuery:
  count = a function that counts
  avg =  computes the average
  sum = computes the sum
  distinct-values = eliminates duplicates
:)


(:                            :)
(::::::::::::::::::::::::::::::)
(:     q15.xq                 :)
(:     Flattening             :)

for $b in doc("bib.xml")/bib/book,
    $x in $b/title,
    $y in $b/author
return <answer>
         { $x,  $y }
       </answer>


(:                            :)
(::::::::::::::::::::::::::::::)
(:     q16.xq                 :)
(:     Regrouping             :)

for $b in doc("bib.xml")/bib
let $y1:= (distinct-values($b/book/year/text())),
    $y2:= (distinct-values($b/book/@year)),
    $y:=($y1, $y2)   (: means concatenation :)
for $x in $y
return
    <answer>
        <year> { $x } </year>
        { for $z in $b/book[(year/text()=$x) or (@year=$
x)]/title
          return $z
        }
    </answer>
```