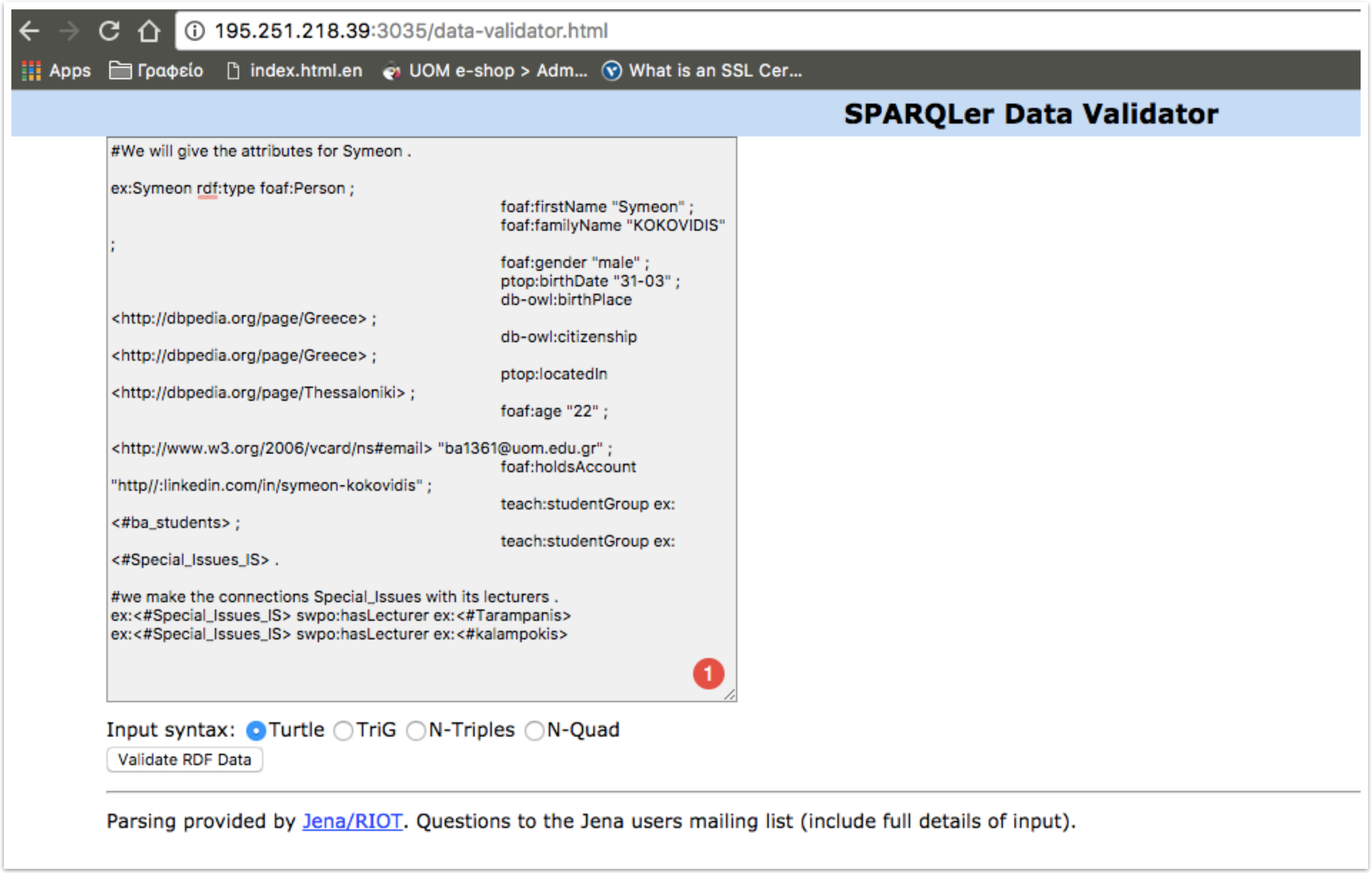
SPARQL Queries

Using our .ttl file we can start by checking for any mistakes on the validator using the following link:
http://195.251.218.39:3035/data-validator.html



```
#We will give the attributes for Symeon .

ex:Symeon rdf:type foaf:Person ;
                                        foaf:firstName "Symeon" ;
                                        foaf:familyName "KOKOVIDIS"
;
                                        foaf:gender "male" ;
                                        ptop:birthDate "31-03" ;
                                        db-owl:birthPlace
<http://dbpedia.org/page/Greece> ;
                                        db-owl:citizenship
<http://dbpedia.org/page/Greece> ;
                                        ptop:locatedIn
<http://dbpedia.org/page/Thessaloniki> ;
                                        foaf:age "22" ;

<http://www.w3.org/2006/vcard/ns#email> "ba1361@uom.edu.gr" ;
                                        foaf:holdsAccount
"http//:linkedin.com/in/symeon-kokovidis" ;
                                        teach:studentGroup ex:
<#ba_students> ;
                                        teach:studentGroup ex:
<#Special_Issues_IS> .

#we make the connections Special_Issues with its lecturers .
ex:<#Special_Issues_IS> swpo:hasLecturer ex:<#Tarampanis>
ex:<#Special_Issues_IS> swpo:hasLecturer ex:<#kalampokis>
```

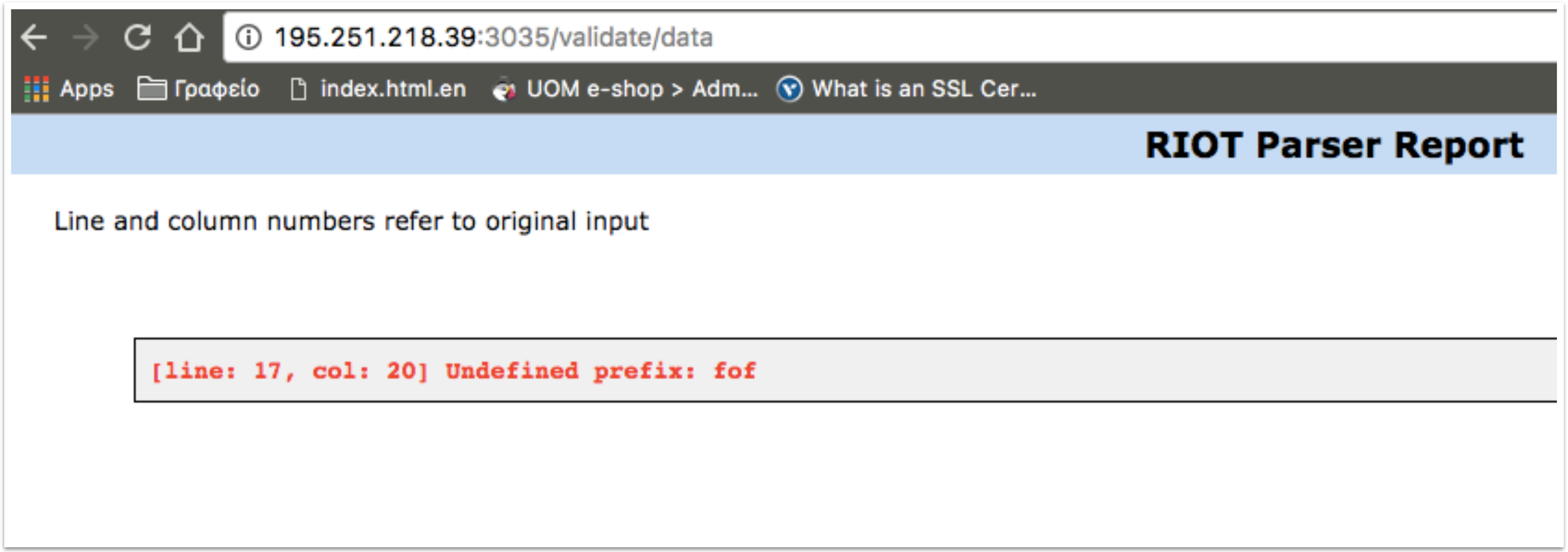
Input syntax: [✓ Turtle] TriG N-Triples N-Quad

Validate RDF Data

Parsing provided by Jena/RIOT. Questions to the Jena users mailing list (include full details of input).

SPARQL Queries

To detect any mistypes or possible faults in the structure of our file.
(e.g. a mistyped prefix ~~fof:Person~~ ->foaf:Person)



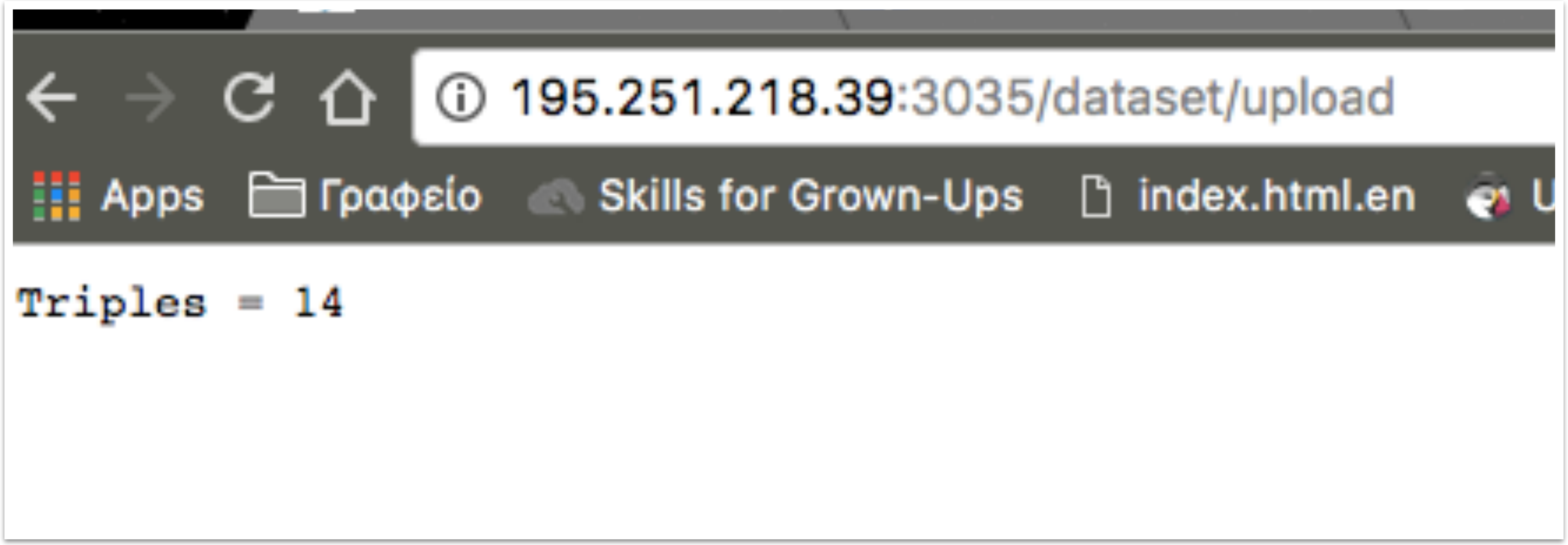
RIOT Parser Report

Line and column numbers refer to original input

```
[line: 17, col: 20] Undefined prefix: fof
```

SPARQL Queries

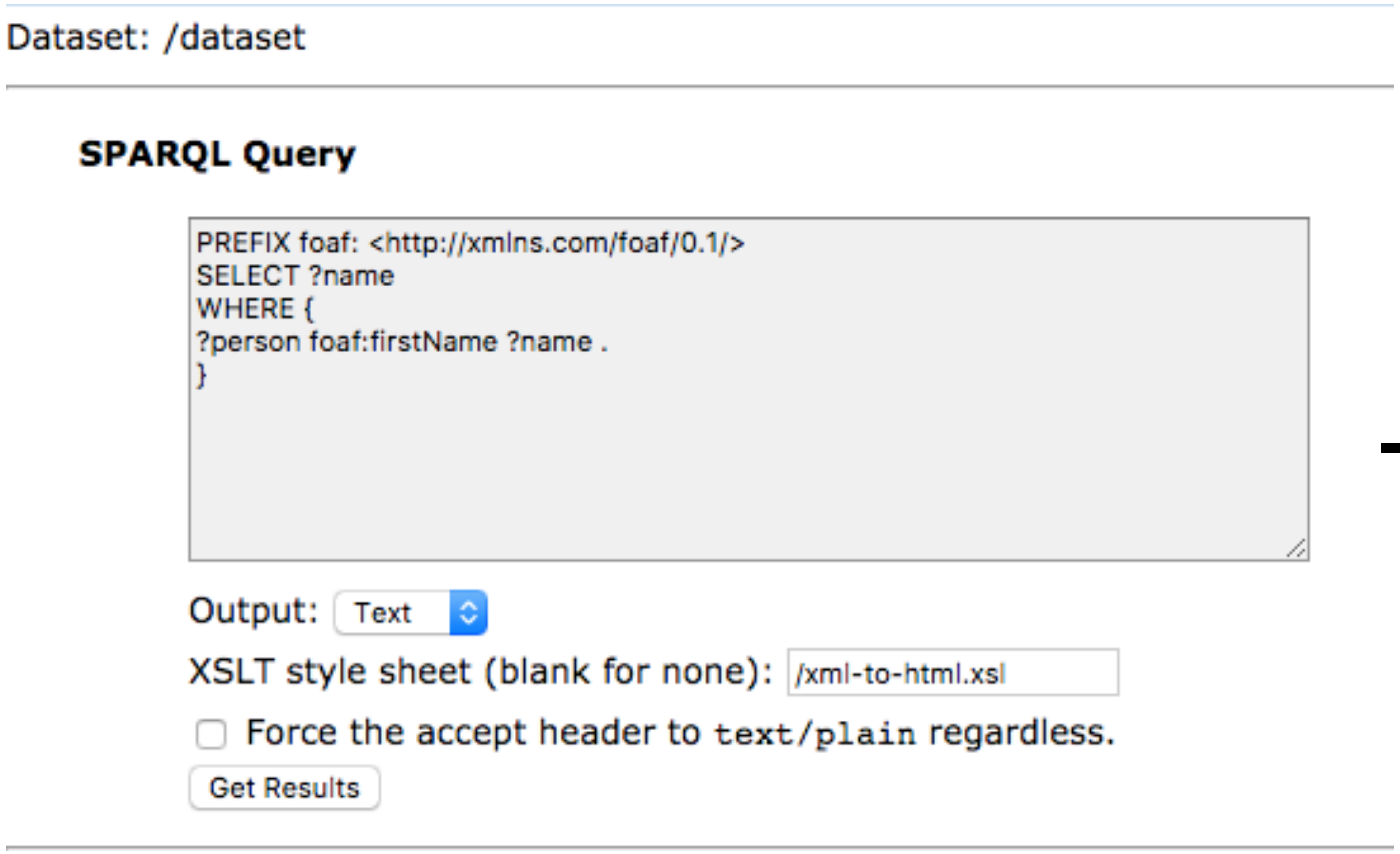
After this check, we are can upload our .ttl file. We confirm the successful upload, as the system returns a message with the number of triples inserted.



```
Triples = 14
```

SPARQL Queries

Now we are ready to start our queries: First, we will try to find all of our partners using the foaf:firstName to fetch all the registered names. The results can be found on the right column



Dataset: /dataset

SPARQL Query

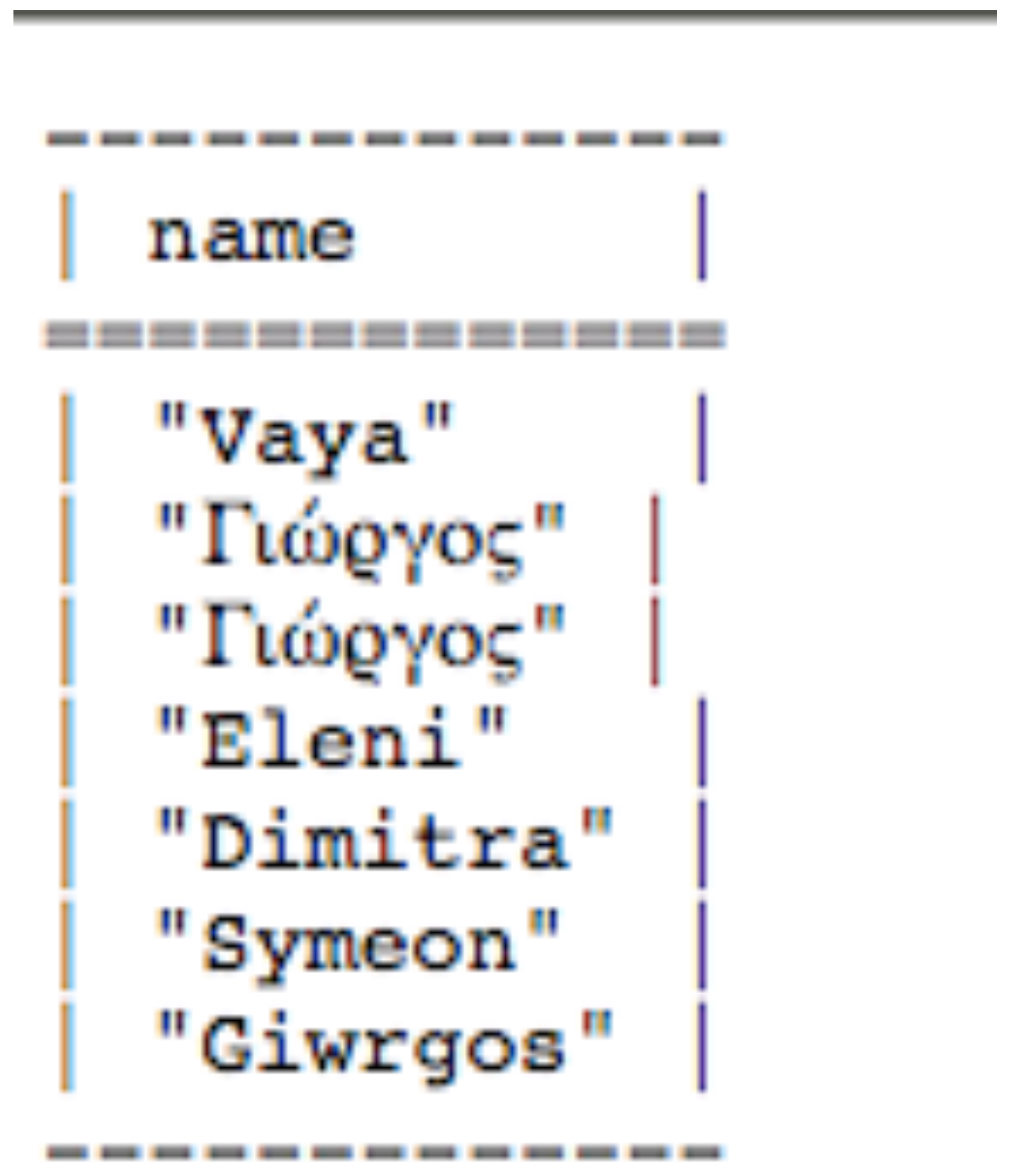
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
?person foaf:firstName ?name .
}
```

Output: Text

XSLT style sheet (blank for none): /xml-to-html.xsl

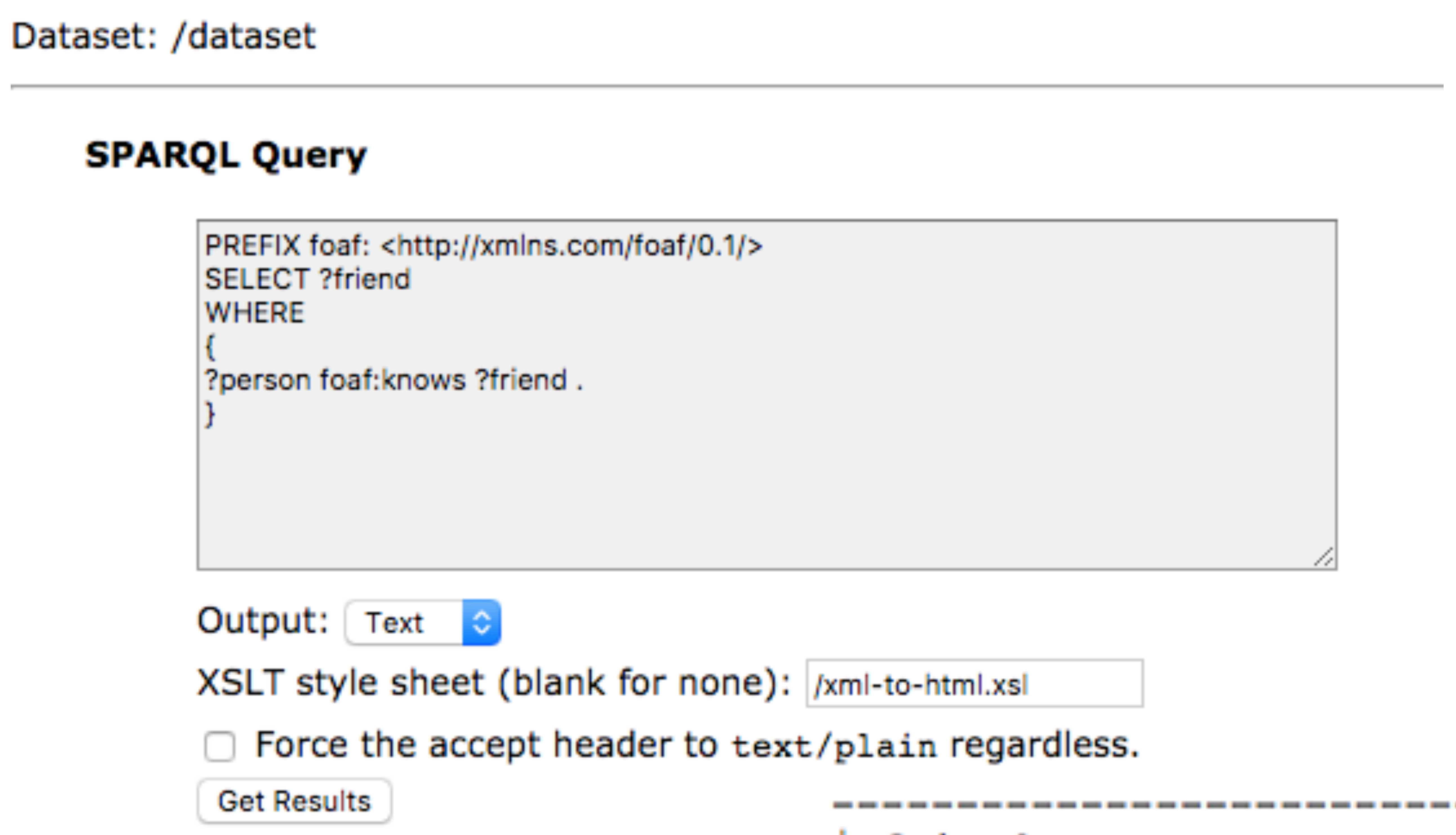
☐ Force the accept header to text/plain regardless.

Get Results



```
-------------
| name      |
=============
| "Vaya"    |
| "Γιώργος" |
| "Γιώργος" |
| "Eleni"   |
| "Dimitra" |
| "Symeon"  |
| "Giwrgos" |
-------------
```

Also to find out if there is any entry with a connection to another contact



Dataset: /dataset

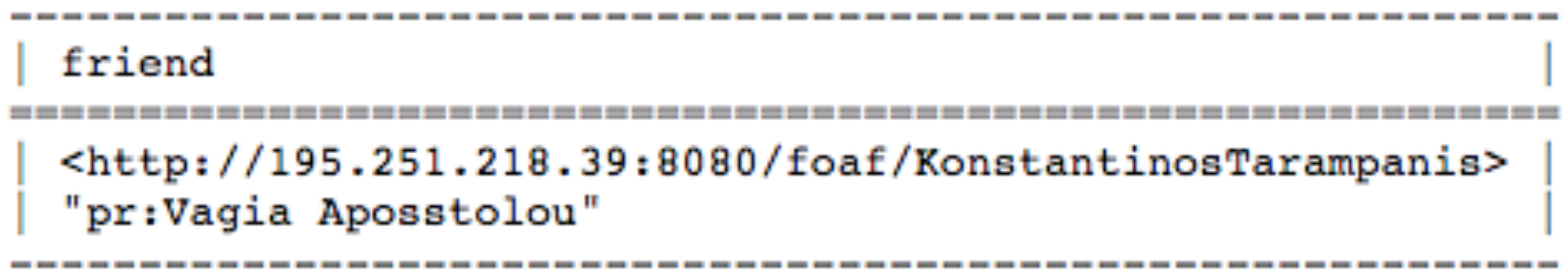
SPARQL Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?friend
WHERE
{
?person foaf:knows ?friend .
}
```

Output: Text

XSLT style sheet (blank for none): /xml-to-html.xsl

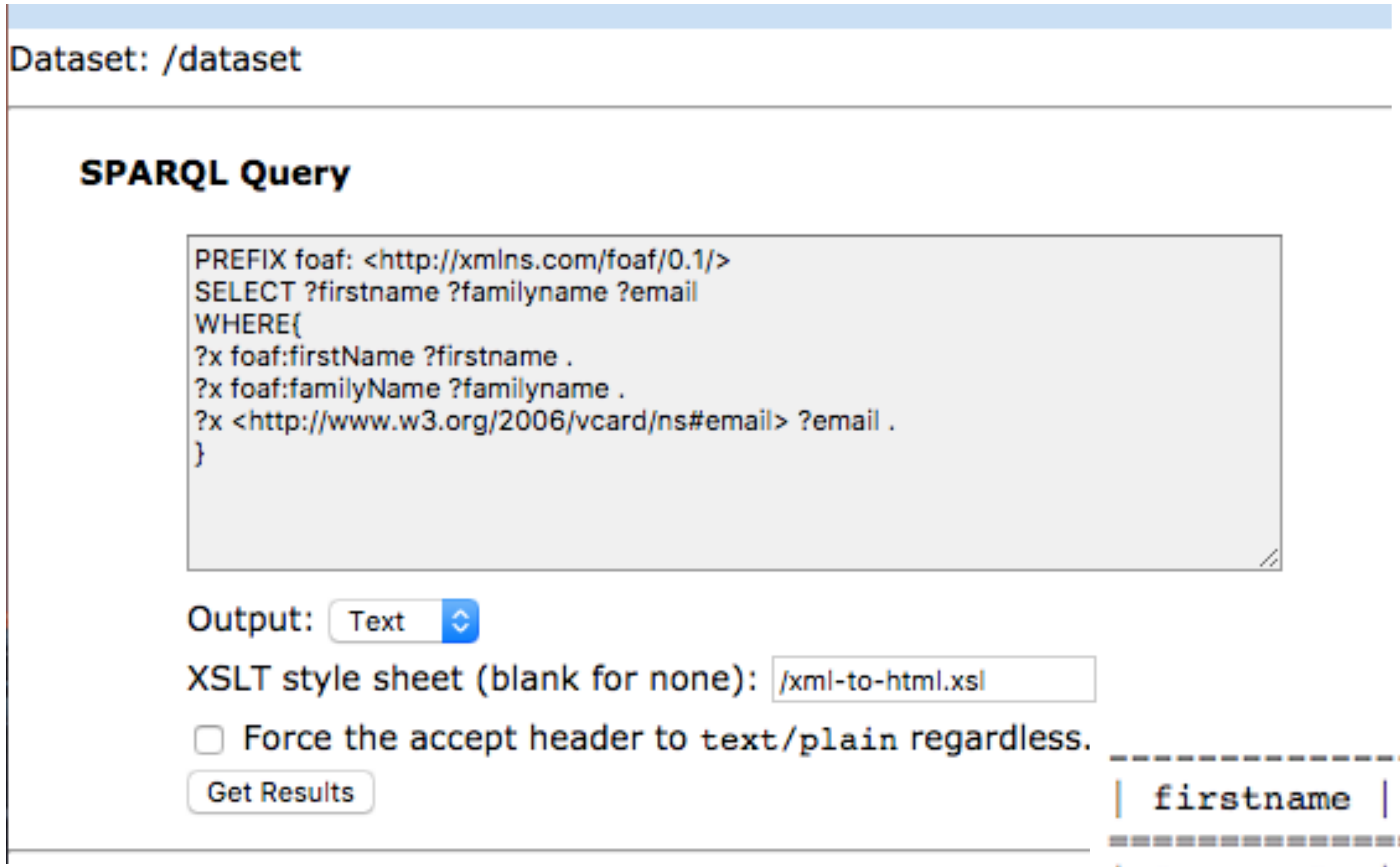
☐ Force the accept header to `text/plain` regardless.



```
--------------------------------------------------------------
| friend                                                     |
==============================================================
| <http://195.251.218.39:8080/foaf/KonstantinosTarampanis>   |
| "pr:Vagia Aposstolou"                                      |
--------------------------------------------------------------
```

SPARQL Queries

In the following query we will try to fetch our e-mail address matched with our first name and family name. Please note that the “firstname” refers to our defined value and there is no restriction on which name to choose. Whereas “firstName” & “familyName” refer to our foaf vocabulary and are pre-defined.



Dataset: /dataset

SPARQL Query

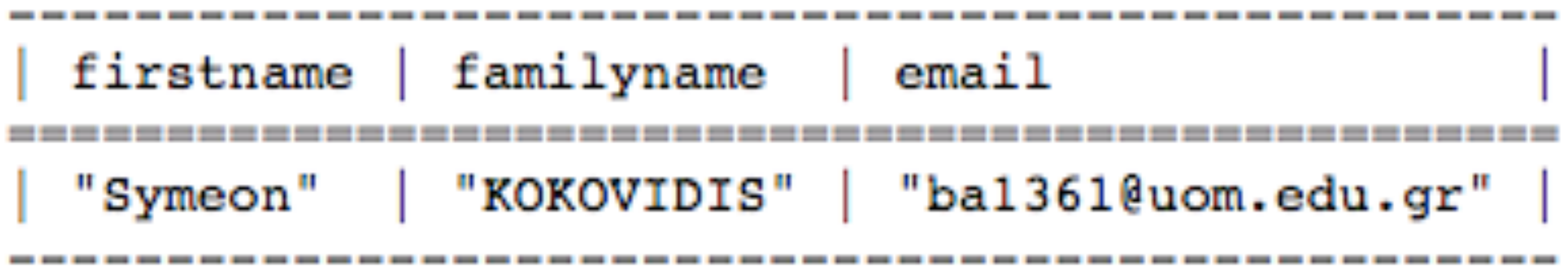
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?firstname ?familyname ?email
WHERE{
?x foaf:firstName ?firstname .
?x foaf:familyName ?familyname .
?x <http://www.w3.org/2006/vcard/ns#email> ?email .
}
```

Output: Text

XSLT style sheet (blank for none): /xml-to-html.xsl

Force the accept header to text/plain regardless.

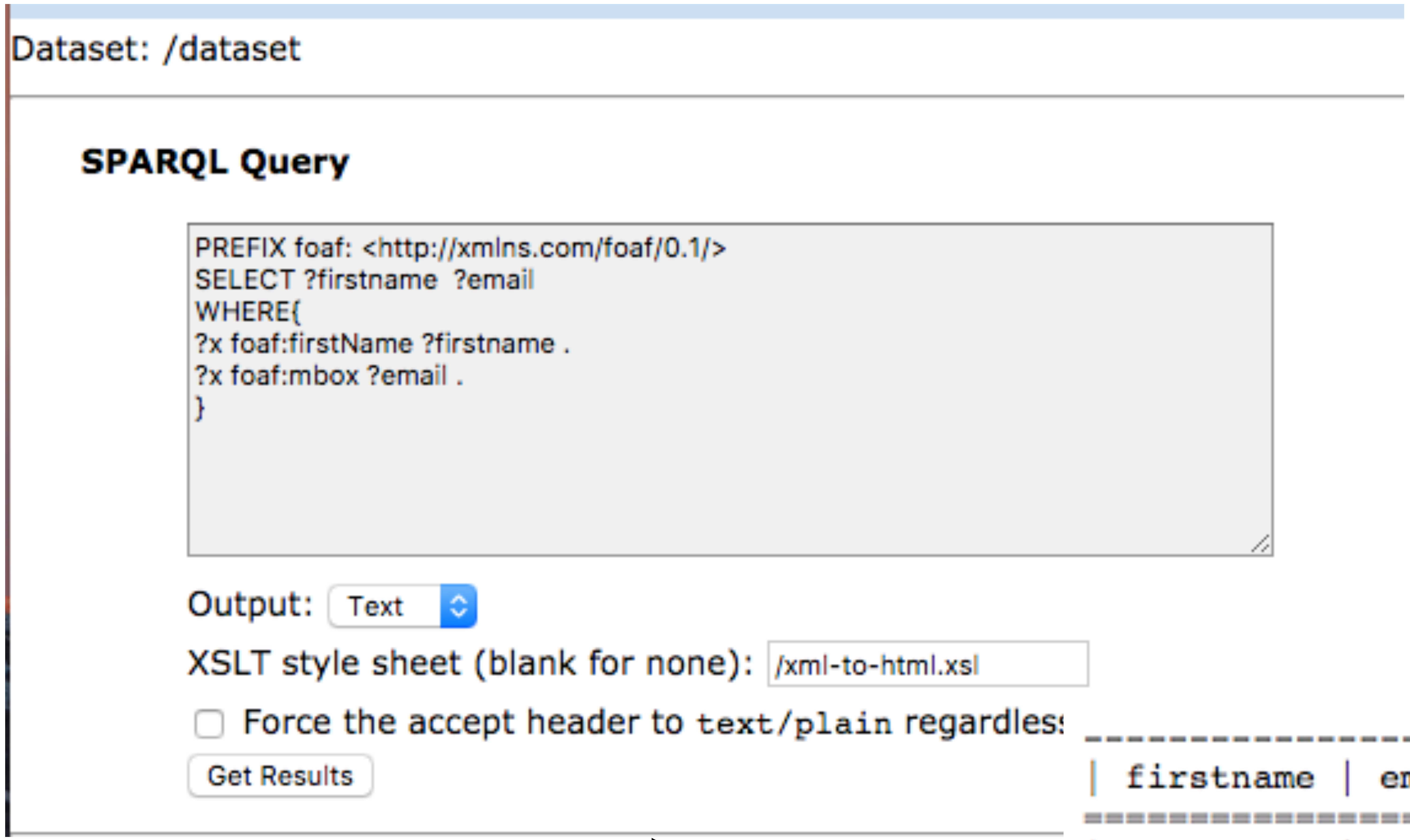
However, in the results we can find only our e-mail address. This might mean we were the only one who chose the vcard vocabulary to define our e-mail address. In the next query we will repeat the request with another vocabulary



```
------------------------------------------------------
| firstname | familyname   | email                   |
======================================================
| "Symeon"  | "KOKOVIDIS" | "ba1361@uom.edu.gr"     |
------------------------------------------------------
```

SPARQL Queries

In accordance with our previous query we now use the foaf vocabulary to fetch the e-mail addresses.



Dataset: /dataset

SPARQL Query

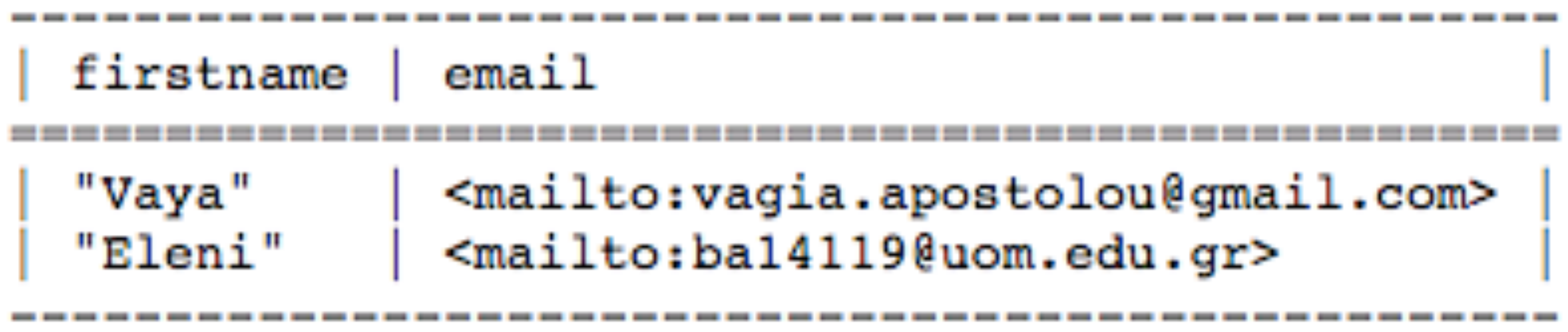
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?firstname  ?email
WHERE{
?x foaf:firstName ?firstname .
?x foaf:mbox ?email .
}
```

Output: Text

XSLT style sheet (blank for none): /xml-to-html.xsl

Force the accept header to text/plain regardles

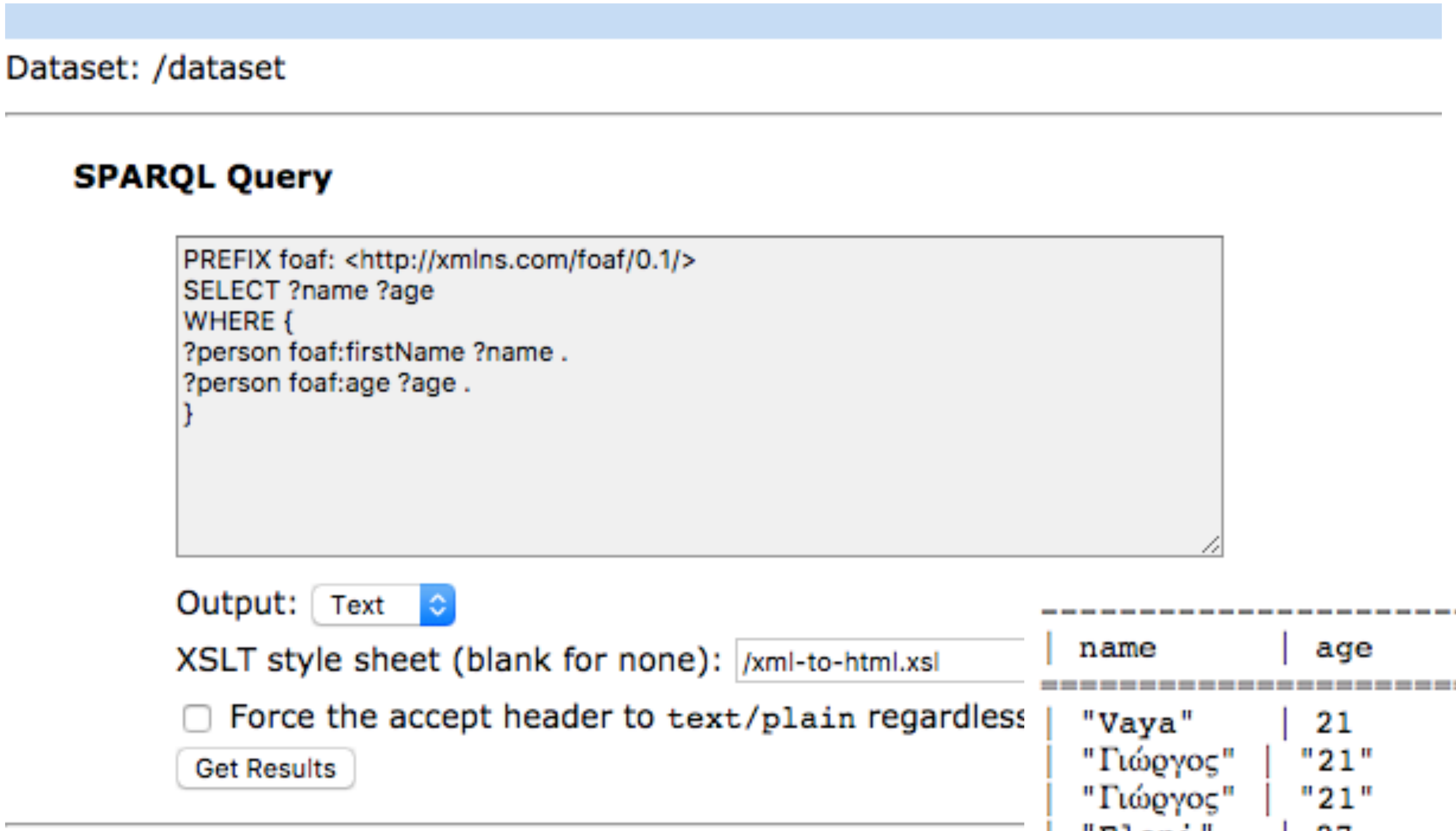
Indeed, we could fetch two more e-mail addresses using a different vocabulary.



```
-------------------------------------------------
| firstname | email                             |
=================================================
| "Vaya"    | <mailto:vagia.apostolou@gmail.com> |
| "Eleni"   | <mailto:ba14119@uom.edu.gr>        |
-------------------------------------------------
```

SPARQL Queries

In our last query we will try to fetch the age of every participant. Then we will try to include a restriction to a second query



Dataset: /dataset

SPARQL Query

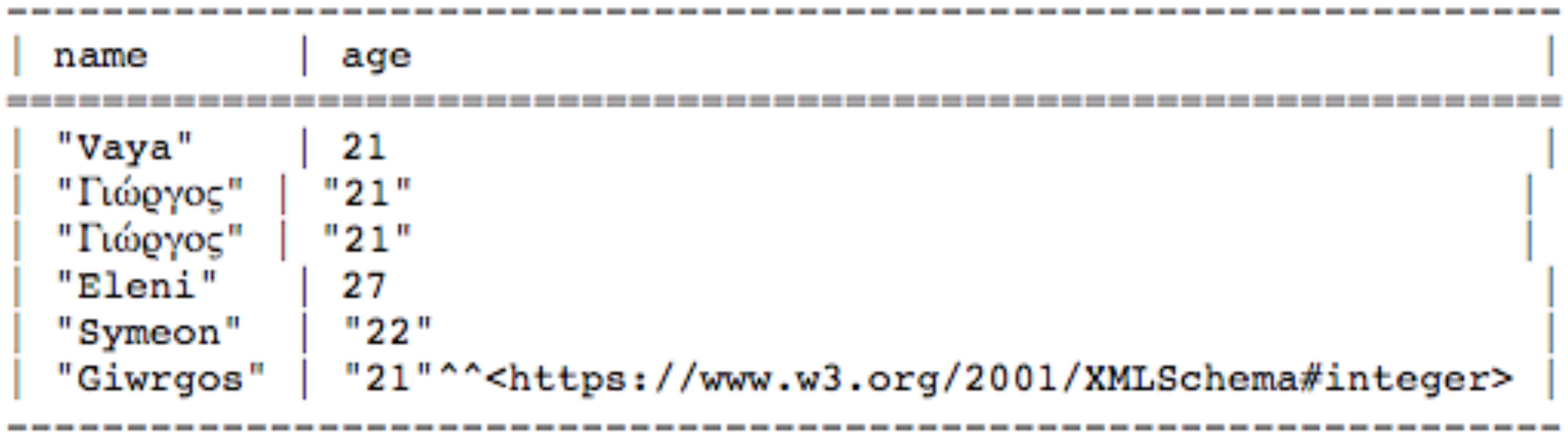
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?age
WHERE {
?person foaf:firstName ?name .
?person foaf:age ?age .
}
```

Output: Text

XSLT style sheet (blank for none): /xml-to-html.xsl

☐ Force the accept header to text/plain regardless

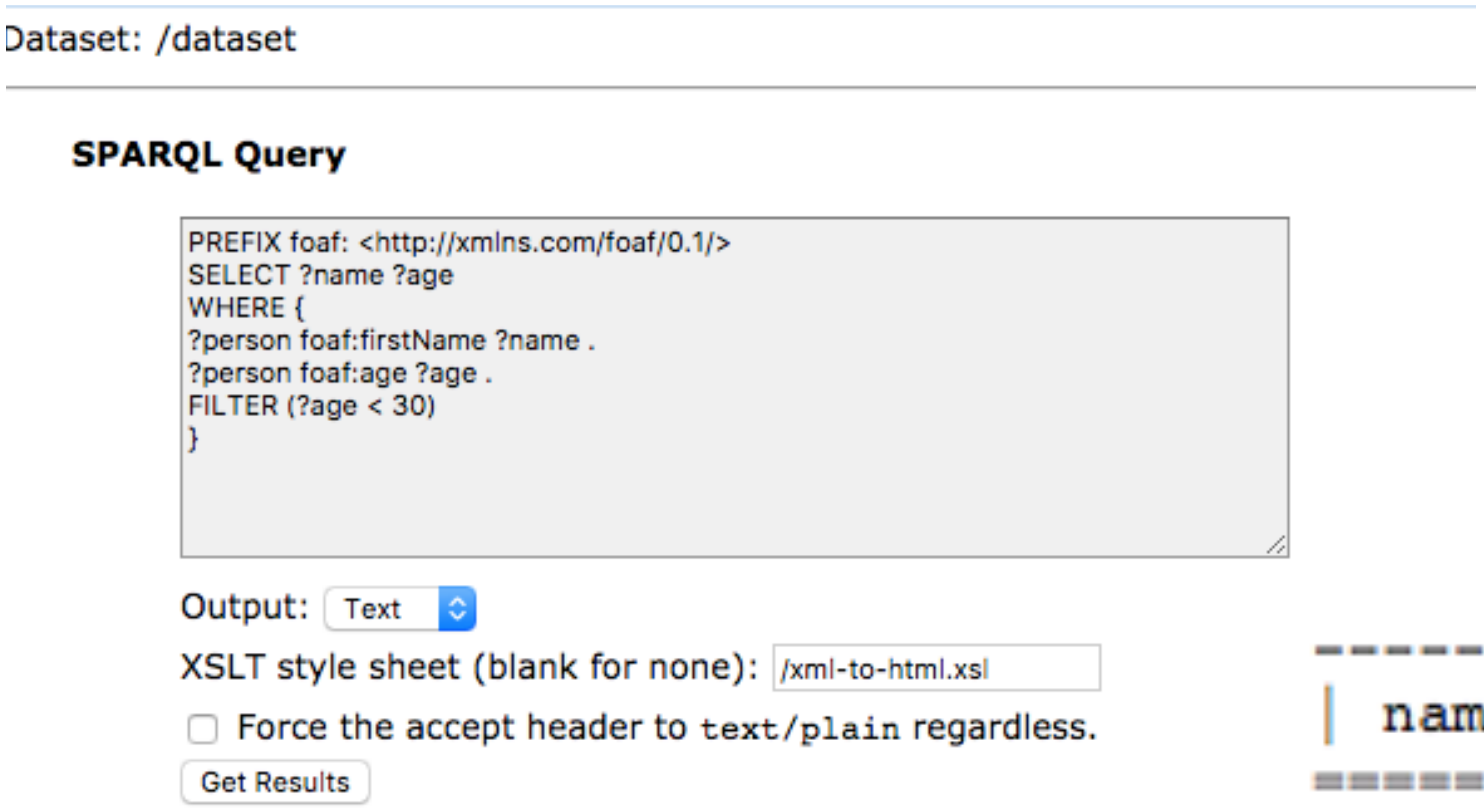
In the results we can find different ages where some of them are registered as a text (included in “”) and other which are registered as numbers (Vaya & Eleni). In case we want to include a restriction, only the entries as numbers will be taken into account.



```
-----------------------------------------------------------------------
| name        | age                                                   |
=======================================================================
| "Vaya"      | 21                                                    |
| "Γιώργος"  | "21"                                                  |
| "Γιώργος"  | "21"                                                  |
| "Eleni"     | 27                                                    |
| "Symeon"    | "22"                                                  |
| "Giwrgos"   | "21"^^<https://www.w3.org/2001/XMLSchema#integer>     |
-----------------------------------------------------------------------
```

SPARQL Queries

Following our previous query we will try to restrict our results. First we will use the FILTER(?age <30) to determine which entries will be used.



Dataset: /dataset

SPARQL Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?age
WHERE {
?person foaf:firstName ?name .
?person foaf:age ?age .
FILTER (?age < 30)
}
```

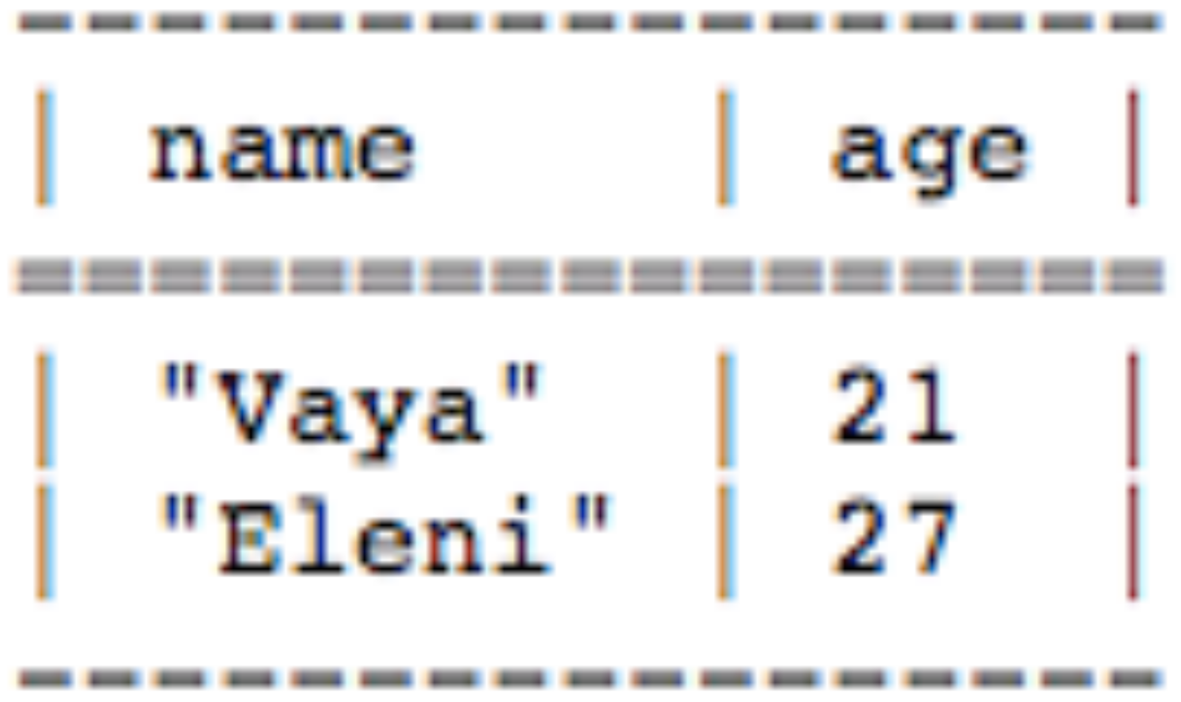
Output: Text

XSLT style sheet (blank for none): /xml-to-html.xsl

☐ Force the accept header to text/plain regardless.

Get Results

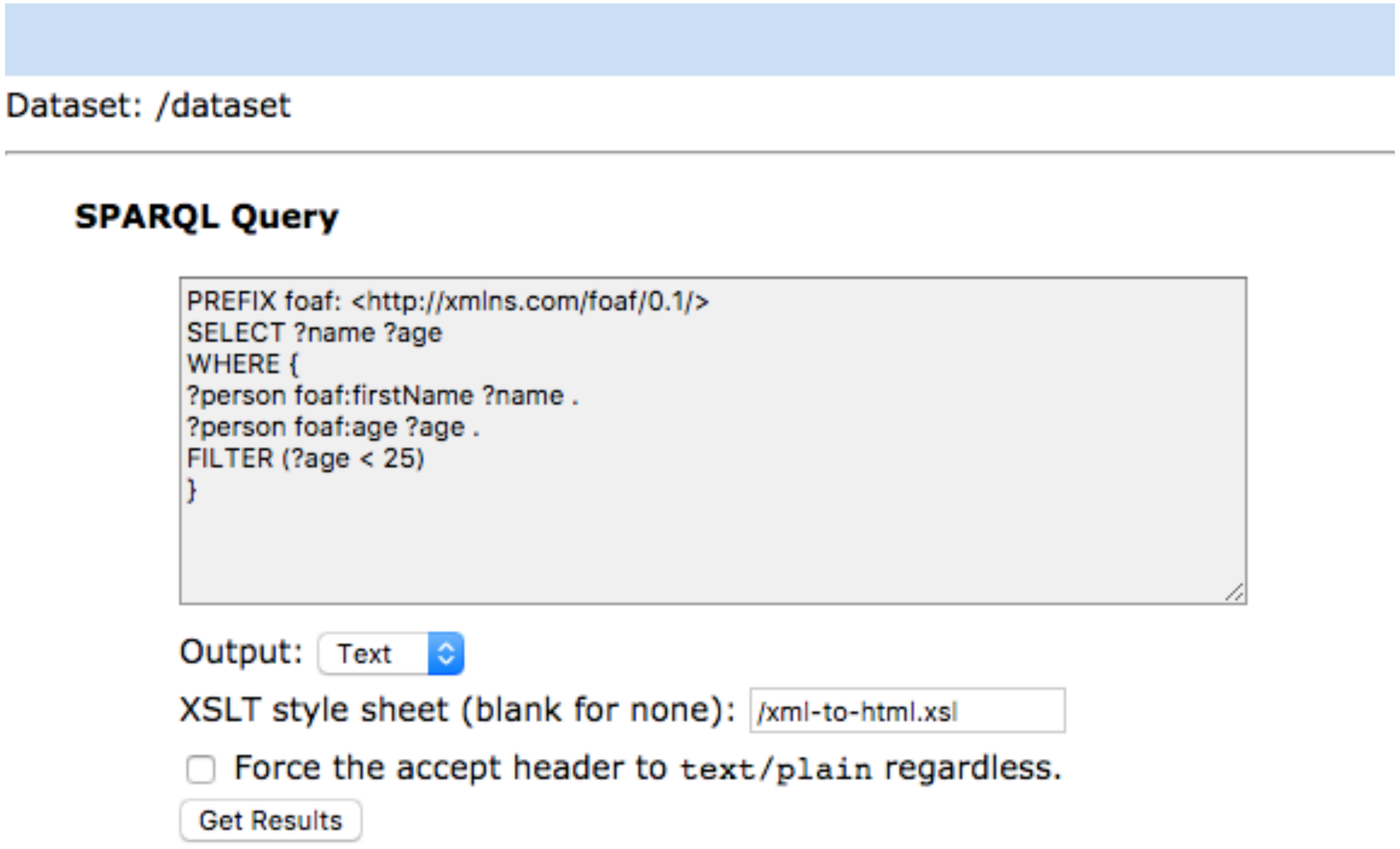
As we previously mentioned, the only entries as a number are from Vaya and Eleni and query returns only these entries. In the next query we will try to fetch only Vaya's age



```
-------------------
| name    | age |
===================
| "Vaya"  | 21  |
| "Eleni" | 27  |
-------------------
```

SPARQL Queries

Following our previous query we will now try to restrict our results with the FILTER(?age <25)



Dataset: /dataset

SPARQL Query

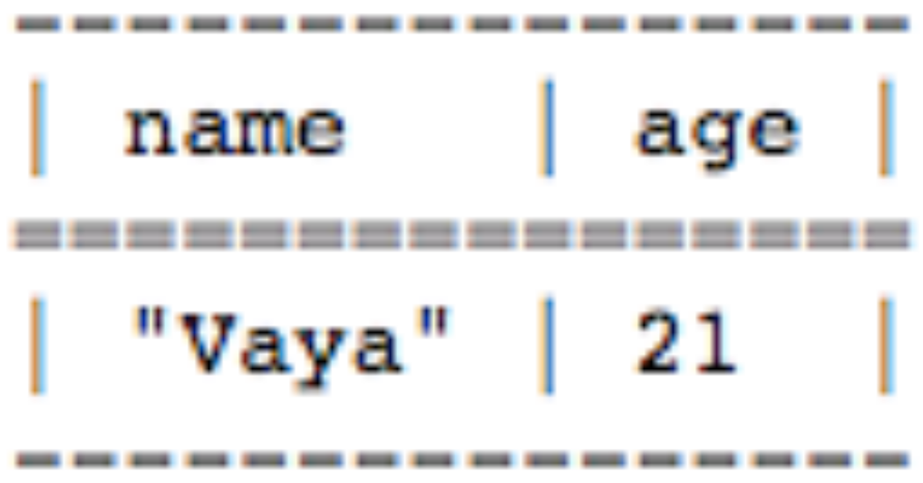
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?age
WHERE {
?person foaf:firstName ?name .
?person foaf:age ?age .
FILTER (?age < 25)
}
```

Output: Text

XSLT style sheet (blank for none): /xml-to-html.xsl

☐ Force the accept header to `text/plain` regardless.

Indeed, the query returned only the entries which have age <25 (and which were registered as a number)



```
-----------------
| name   | age |
=================
| "Vaya" | 21  |
-----------------
```

Remarks

Is really important to use the same vocabularies for well defined attributes. In case of different vocabularies (e.g. foaf / vcard) is unavoidable to miss results.

In case we want to filter our results is advisable to record our data with their correct type. For example numbers should not be recorded as alphabetical data

For the course of Special Issues on Information Systems

Symeon Kokovidis
ba1361@uom.edu.gr
December 2016

Information Systems Lab
University of Macedonia - Greece

HELLENIC REPUBLIC
UNIVERSITY OF MACEDONIA