Using the triples_creator module

To use the module on your code, simply add "import triples_creator"

In the beginning of the code.

The main function of the module is the function create_triples:

```
   def create_triples( url, subject_query, subject_func, predicate,
object_query, object_func)
```

Parameters:

url (string) – the url of page from which you extract the data.

subject_query (string) – xpath query for the subject

subject_func (function pointer) – function pointer for a user defined function for the processing of the subject query input, or 0 If no process necessary.

predicate (string) – string representing the predicate.

object_query (string) – xpath query for the subject. The query should be relative to the subject query. In case no query required for the subject will be 0.

object_func (function pointer) – function pointer for a user defined function for the processing of the object query input, or 0 If no process necessary.

Return value:

The function returns a list of triples. A triple is a dictionary element with the keys: 'subject', 'predicate', 'object'.

Example of using:

We will extract triples from the faculty page of the Technion computer science department site. http://www.cs.technion.ac.il/people/faculty/.

From the page html code, we will extract the name and homepage of each faculty member. The executing code is on faculty_example.py. The relevant information in the html code of the html page is inside the element marked as `<ul class="peoplelist">,` in lines 85-254.

First, we will define a URI for each person to be the URI of his faculty page. The URIs are located for each person in the second "a" element, inside the "href" property:

```
<li><a href="http://nailon.googlepages.com/"
title="Homepage of Nir Ailon"
class="homelink"><span>[homepage]</span></a><a
href="/people/nailon/" title="Department info page
for Nir Ailon">Ailon, Nir</a></li>
```

We will define the xpath query to extract all the URIs:

```
subject_query_person='//ul[@class="peoplelist"]/li/a[last()]/@href'
```

And this will be the subject query. We want to get the full URI for each person, so we will add to the query outputs the prefix http://www.cs.technion.ac.il/. We will also add angle brackets around the URI. Thus, the processing function for the subject query output will be:

```
def subject_func_person(s):
    return '<http://www.cs.technion.ac.il'+s+'>'
```

We want to define these URIs as persons, so the predicate is rdf:type, And the object is foaf:Person. We define:

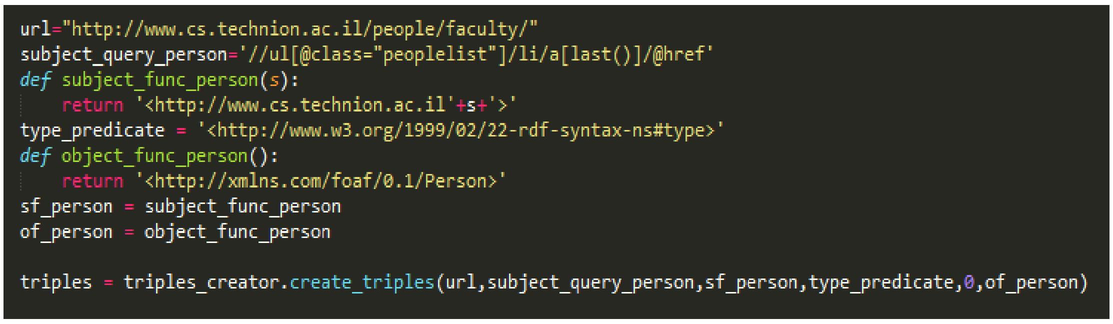
```
type_predicate = '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
```

And since in this case the object is constant, we don't need query for the object, so we set the object query to be 0, and the processing function for the object will be:

```
def object_func_person():
    return '<http://xmlns.com/foaf/0.1/Person>'
```

We define the parameters, and now we can run the function to get

The triples:



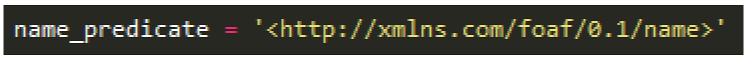
```
url="http://www.cs.technion.ac.il/people/faculty/"
subject_query_person='//ul[@class="peoplelist"]/li/a[last()]/@href'
def subject_func_person(s):
    return '<http://www.cs.technion.ac.il'+s+'>'
type_predicate = '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
def object_func_person():
    return '<http://xmlns.com/foaf/0.1/Person>'
sf_person = subject_func_person
of_person = object_func_person

triples = triples_creator.create_triples(url,subject_query_person,sf_person,type_predicate,0,of_person)
```

Now we want to get for each person's URI the matching person's name.

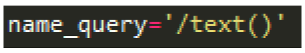
```
<li><a href="http://nailon.googlepages.com/"
title="Homepage of Nir Ailon"
class="homelink"><span>[homepage]</span></a><a
href="/people/nailon/" title="Department info page
for Nir Ailon">Ailon, Nir</a></li>
```

The subjects are again the URIs, so the subject query and the processing function will be the same as in the previous run (and obviously the page url is also the same). The predicate will be foaf:name:



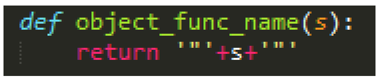
```
name_predicate = '<http://xmlns.com/foaf/0.1/name>'
```

We now need to define the xpath query for the object. The query Is relative to the subject, thus we need to navigate from the element of the subject (the URI) to the element of the object (the name), and get the data. In this case, the URI and the name are in the same "a" element, Therefore our query remain in the current element and extract the text of the element using the text() function:



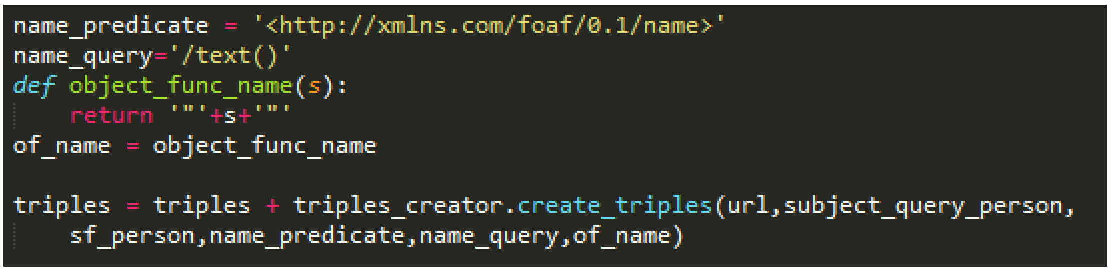
```
name_query='/text()'
```

We want to add quotation marks to the name, thus are processing function will be:



```
def object_func_name(s):
    return '"'+s+'"'
```

Now we can run create_triples again and the triples received to the previous triples:



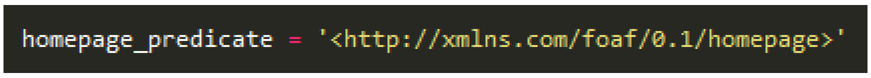
```python
name_predicate = '<http://xmlns.com/foaf/0.1/name>'
name_query='/text()'
def object_func_name(s):
    return '"'+s+'"'
of_name = object_func_name

triples = triples + triples_creator.create_triples(url,subject_query_person,
    sf_person,name_predicate,name_query,of_name)
```

Now we want to get for each person's URI the matching person's homepage.

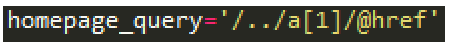
```
<li><a href="http://nailon.googlepages.com/"
title="Homepage of Nir Ailon"
class="homelink"><span>[homepage]</span></a><a
href="/people/nailon/" title="Department info page
for Nir Ailon">Ailon, Nir</a></li>
```

Again, no change in the subject query and the processing function. The predicate will be foaf:homepage:



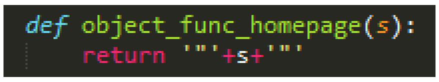
```python
homepage_predicate = '<http://xmlns.com/foaf/0.1/homepage>'
```

Now, we define the xpath query from the URI to the homepage. We need to get from the second "a" element to the first "a" element, and then extract the "href" propertie. We define:



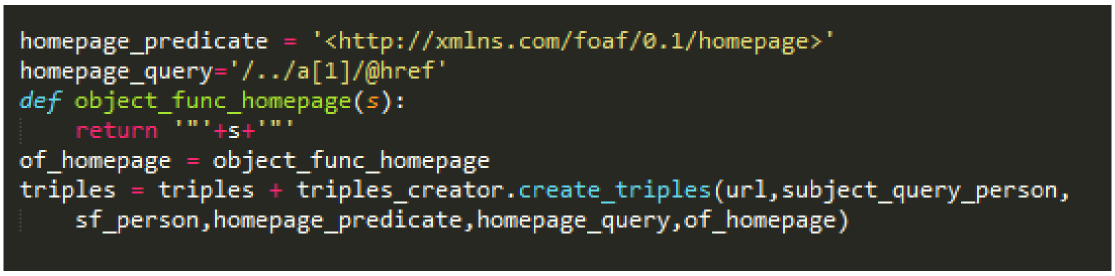
```python
homepage_query='/../a[1]/@href'
```

We want to add quotation marks; thus the processing function will be:



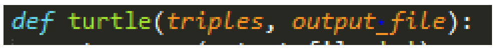
```python
def object_func_homepage(s):
    return '"'+s+'"'
```

And we run create_triples again to add our new triples:



```python
homepage_predicate = '<http://xmlns.com/foaf/0.1/homepage>'
homepage_query='/../a[1]/@href'
def object_func_homepage(s):
    return '"'+s+'"'
of_homepage = object_func_homepage
triples = triples + triples_creator.create_triples(url,subject_query_person,
    sf_person,homepage_predicate,homepage_query,of_homepage)
```

We now have our list of triples. We can now turn it to rdf file using the function turtle, which create an rdf in turtle format:

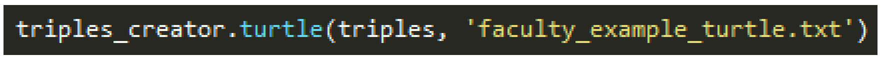


```
def turtle(triples, output_file):
```

Parameters:

triples – list of triples.

output_file – name of the output file.



```
triples_creator.turtle(triples, 'faculty_example_turtle.txt')
```

We now have our data in turtle format in the file faculty_turtle_example.txt.