
QApedia

Versão v0.0.0-alpha

03 jul, 2019

1	Introdução	3
1.1	O que é o QApedia?	3
2	Tutorial	7
2.1	Introdução	7
3	Guia de Usuário	11
3.1	Instalação	11
3.2	Carregando os templates	11
3.3	Ajustando a <i>query geradora</i>	12
3.4	Realizando uma consulta	13
3.5	Geração de pares	13
4	QApedia package	17
4.1	Módulos	17
4.2	QApedia.qapedia module	17
4.3	QApedia.generator module	18
4.4	QApedia.io module	22
4.5	QApedia.utils module	23
	Índice de Módulos do Python	27
	Índice	29

O QApedia é uma ferramenta de geração de pares de questão-sparql escrito em Python a partir do endpoint previamente estabelecido. O seu nome vem da junção dos termos QA de *Question-Answering* e *pedia* da palavra wikipedia. O repositório contendo o código fonte pode ser encontrado em [QApedia/QApedia](#) e alguns exemplos estão disponíveis na pasta [exemplos](#) do respectivo repositório.

1.1 O que é o QApedia?

QApedia («QA» de Question-Answering, «pedia» de Wikipédia) é um pacote escrito em python que tem como objetivo realizar a geração de pares de questões-queries com base em um template previamente estabelecido sobre o endpoint da **DBpedia**. Esses pares gerados podem ser utilizado na criação de um sistema de *Question-Answering (QA)*, onde a pergunta é fornecida em linguagem natural e a sua resposta seria o resultado da consulta Sparql sobre a base de dados da DBpedia.

1.1.1 DBpedia

A **DBpedia** («DB» de database) é um projeto criado pela comunidade que cria e fornece acesso aos dados estruturados extraídos da Wikipédia. Esses dados possuem conexões de modo que consultas possam ser realizadas em cima da Wikipédia similarmente a consultas realizadas em banco de dados, esses dados também são conhecidos como *Linked Open Data*. Tim Berners-Lee listou quatro princípios da *Linked Data* em sua nota [Linked Data](#), são eles:

1. Use [Uniform Resource Identifier](#) (URIs) para nomear para coisas
2. Use HTTP URIs para que as pessoas possam procurar esses nomes.
3. Quando alguém procura um URI, forneça informações úteis, usando os padrões (RDF*, SPARQL)
4. Inclua links para outros URIs. Para que eles possam descobrir mais coisas.

Na imagem a seguir é mostrada a página que representa uma entidade do tipo evento, a entidade no caso é [Balaiada](#) que possui uma identificação URI, além de um conjunto de informações agregadas, como o local onde ocorreu esse evento (`dbo:place`) ou um breve resumo sobre esse evento (`dbo:abstract`).

About: Balaiada

An Entity of Type : [event](#), from Named Graph : <http://dbpedia.org>, within Data Space : [dbpedia.org](#)

The Balaiada was a social revolt that occurred between 1838 and 1841 in the interior of the province of Maranhão, Brazil.

Property	Value
dbpedia:abstract	The Balaiada was a social revolt that occurred between 1838 and 1841 in the interior of the province of Maranhão, Brazil. ^(en)
dbpedia:combatant	- Rebels *15pxNational Guard *15pxImperial Navy - Empire of Brazil *African slaves *Balaios
dbpedia:commander	dbpedia:Luis_Alves_de_Lima_e_Silva dbpedia:Cosme_Bento dbpedia:Raimundo_Gomes
dbpedia:place	dbpedia:Brazil
dbpedia:result	Legalist victory
dbpedia:strength	8,000 10,000-12,000
dbpedia:thumbnail	wiki-commons:Special:FilePath/Tropas_brasileiras_1835.jpg?width=300
dbpedia:wikiPageID	12900447 ^(xsd:integer)
dbpedia:wikiPageRevisionID	733591416 ^(xsd:integer)
dbpedia:caption	Troops departing from Rio de Janeiro to Maranhão.
dbpedia:date	1838 ^(xsd:integer)
dbpedia:subject	dbpedia:Maranhão dbpedia:19th-century_conflicts dbpedia:Rebellions_in_Brazil

1.1.2 Dados RDF

O **Resource Description Framework (RDF)** é um framework que serve para expressar informações sobre recursos. Qualquer coisa pode ser considerada um recurso, seja ela um conceito abstrato, documentos, pessoas ou objetos físicos, por exemplo. O RDF permite realizar declarações sobre os recursos sempre seguindo a seguinte estrutura:

<sujeito><predicado><objeto>

Através de uma instrução RDF, os recursos (sujeito e objeto) são relacionados através de um predicado que representa a natureza desse relacionamento. Sendo esse relacionamento direcional (do sujeito para o objeto) nomeado propriedade em RDF. A seguir, um exemplo de triplas RDF representado em pseudocode do recurso **Balaiada**.

<Balaiada> <é um> <evento>.

<Balaiada> <é a história das> <Forças Armadas Brasileiras>.

<Balaiada> <foi em> <1838>.

<Balaçada> <foi no> <Brasil>.

<Balaçada> <teve como resultado> <vitória legalista>

O conjunto de dados RDF da DBpedia é hospedado e publicado usando o [OpenLink Virtuoso](#). Através da infraestrutura fornecida pelo Virtuoso é possível acessar os dados RDF da DBpedia através do endpoint SPARQL usando GET padrão de cliente WEB HTTP. Esse endpoint é utilizado na realização de buscas SPARQL realizadas pelo pacote `qapedia` para recuperar esses dados da Wikipedia.

2.1 Introdução

Necessita de um conjunto de questões-sparql? Com o *QApedia* vamos mostrar como realizar essa tarefa de forma a ajudá-lo nesse problema de construção desse dataset.

Este tutorial explica como funciona o processo de geração dos pares de questão-sparql através do pacote *QApedia*. Explicando como o arquivo de entrada deve estar estrutura até o resultado gerado. Primeiramente vamos definir algumas informações importantes utilizadas nesse documento.

- **SPARQL Endpoint** - URL onde é possível realizar consultas SPARQL com resultados via HTTP.
- **question** (pergunta) - pergunta em linguagem natural
- **query** - resposta da pergunta em formato de SPARQL
- **lacuna** (placeholder) - campo presente tanto na *pergunta* quanto na *query* configurando uma pergunta/query genérica.
- **generator query** (query geradora) - *query* que gera as possíveis opções que podem ser utilizadas para preencher as *lacunas*.
- **template** - Estrutura contendo os elementos *pergunta*, *query* e *generator query*
- **rdfs:label** - usada para fornecer uma versão legível do nome de um recurso.
- **URI** - Identificador de um recurso abstrato ou físico. Exemplos:

```
http://dbpedia.org/resource/Hunter_%C3%97_Hunter  
https://pt.wikipedia.org/wiki/Hypertext_Transfer_Protocol  
https://qapedia.readthedocs.io/pt/latest/
```

- **Prefix** - São abreviações de URIs. São similares aos *imports* e *includes*, onde são definidas as abreviações utilizadas.
- **resource** (recurso) - são representados com URIs, e podem ser abreviados como nomes prefixados.

Uma consulta SPARQL geralmente pode ser representada da seguinte forma.

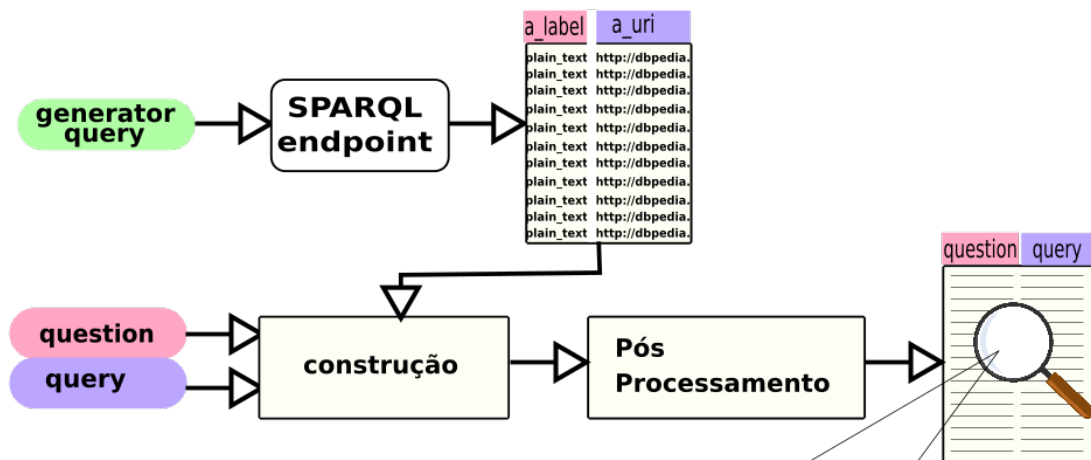
```

# prefix declarations
# nomeia uma URI de forma a abreviar
PREFIX foo: <http://example.com/resources/>
...
# dataset definition
# Indica os grafos RDFS consultados
FROM ...
# result clause
# Indica a informação a ser retornada da consulta
SELECT ...
# query pattern
# Especifica o que consultar no conjunto de dados
WHERE {
    ...
}
# query modifiers
# reorganiza os dados da consulta
# pode limitar a quantidade de resultados, reordenar, etc.
ORDER BY ...

```

Na Figura abaixo temos o processo de geração do conjunto de pares de questão-sparql, onde desejamos obter o formato de *pergunta* «Fulano é autor de que?» e a *query* «select ?obra where { ?obra tem_autor Fulano}». O *placeholder* é definido no formato <LETRA> e está presente na *question* e *query*.

Template	
question	<A> é autor de que?
query	SELECT DISTINCT ?uri where { ?uri <http://dbpedia.org/ontology/author> <A> }
generator query	select distinct ?a where { ?uri <http://dbpedia.org/ontology/author> ?a }



Exemplo

George R. R. Martin é autor de que?

```
select distinct ?uri where { ?uri <http://dbpedia.org/ontology/author> <http://dbpedia.org/resource/George_R._R._Martin> }
```

A *generator_query* é utilizada para realizar uma consulta no endpoint especificado e ajustada para retornar os campos de *rdfs:label* e a *URI* do recurso autor. Com esse resultado, preenchemos as lacunas presente no template e geramos pares similares ao exemplo mostrado. Nas próxima seção é mostrado o guia de usuário contendo instruções de uso das

funções do pacote. Alguns [exemplos](#) estão disponíveis no repositório do QApedia.

3.1 Instalação

A priori, para se utilizar o pacote do QApedia é necessário realizar a instalação do mesmo. A instalação pode ser executada de dois modos:

3.1.1 Através das releases disponíveis no Github

Em [Releases](#) há a versão compactada .zip e .tar.gz do código. No terminal, caso escolha o formato .tar.gz, no ubuntu, podem ser realizadas as seguintes operações:

```
foo@bar:~$ wget https://github.com/QApedia/QApedia/archive/0.2.2.tar.gz
foo@bar:~$ tar -xvzf 0.2.2.tar.gz
foo@bar:~$ cd QApedia-0.2.2/
foo@bar:~/QApedia-0.2.2$ pip install .
```

3.1.2 Através do github

É possível instalar a versão mais atual disponível no Github através do seguinte comando em seu terminal. É necessário possuir o `_git_` instalado.

```
foo@bar:~$ pip install git+https://github.com/QApedia/QApedia.git
```

3.2 Carregando os templates

Carregue o pacote para poder utilizar suas funcionalidades

```
>>> import QApedia
```

Você pode carregar o seu dataset utilizando a função `QApedia.io.load_templates()` presente no pacote. O arquivo do dataset deve estar no formato csv, a primeira linha deve conter o nome das colunas. Esses nomes devem ser os mesmos que são mostrados no [Tutorial](#). São eles:

- **query** - consulta SPARQL contendo uma lacuna no formato `<LETTER>`.
- **question** - pergunta em linguagem natural contendo a mesma lacuna presente na *query*.
- **generator_query** - query utilizada para preencher as lacunas e assim permitir a geração do conjunto de question-query.

Na tabela seguir é mostrado um exemplo de dataset contendo apenas um template.

question	query	generator_query
<A> é autor de que?	select distinct ?uri where { ?uri dbo:author <A> }	select distinct ?a where { ?uri dbo:author ?a }

O template é carregado como um `pandas.DataFrame`, então as operações de Dataframe podem ser realizadas em cima do conjunto de dados. As variáveis presentes no select da **generator_query** são extraídas no processo de leitura do conjunto de dados e são disponibilizadas na coluna **variables**. No exemplo a seguir, podemos ver as primeiras linhas do arquivo `templates.csv`.

```
>>> from QApedia import io
>>> templates = io.load_templates("templates.csv")
>>> templates.head()
               question  ... variables
0  <A> e <B> é produzido por qual empresa?  ...  [a, b]
1  <A> e <B> é o trabalho notável de qual autor?  ...  [a, b]
2  <A> e <B> são escritos por qual autor?  ...  [a, b]
3  <A> escreveu qual livro?  ...  [a]
4  <A> pertence a qual partido político?  ...  [a]

[5 rows x 4 columns]
```

Para o exemplo mostrado na tabela anterior, sobre a **generator_query** «`select distinct ?a where { ?uri dbo:author ?a }`», a variável extraída seria o *a*.

3.3 Ajustando a query geradora

Uma *generator_query* possui o seguinte formato:

```
#<A> e <B> são os trabalhos notáveis de qual escritor?
select distinct ?a ?b where {
    ?uri <http://dbpedia.org/property/notableworks> ?a .
    ?uri <http://dbpedia.org/property/notableworks> ?b .
    ?uri a <http://dbpedia.org/ontology/Writer>
}
```

Onde *a* e *b* correspondem as lacunas `<A>` e `<B>` definidas na *questão* e *query* do respectivo template. Nesse exemplo, a consulta irá retornar uma lista contendo os URIS dos trabalhos notáveis *a* e *b* de cada escritor. Entretanto, para preencher a lacuna da pergunta em linguagem natural, precisamos do nome desses recursos em linguagem natural, para isso, são extraídas a propriedade **rdfs:label** de cada uma dessas URIs. As *generator_query* definida em nosso template não possui esses campos, então para isso pode ser utilizada a função `QApedia.generator.adjust_generator_query()` que irá inserir as variáveis *la* e *lb* que correspondem as *labels* extraídas sobre cada recurso.

```
>>> from QApedia import generator
>>> generator_query = "select distinct ?a ?b where {\"\\
...           \"?uri <http://dbpedia.org/property/notableworks> ?a . \"\\
...           \"?uri <http://dbpedia.org/property/notableworks> ?b . \"\\
...           \"?uri a <http://dbpedia.org/ontology/Writer> }\"
>>> variables = [\"a\", \"b\"]
\"select distinct ?a ?b ?la ?lb where {?a rdfs:label ?la. FILTER(lang(?la) = 'pt'). ?b
↪rdfs:label ?lb. FILTER(lang(?lb) = 'pt'). ?uri <http://dbpedia.org/property/
↪notableworks> ?a . ?uri <http://dbpedia.org/property/notableworks> ?b . ?uri a
↪<http://dbpedia.org/ontology/Writer> }\"
```

3.4 Realizando uma consulta

Uma busca pode ser feita utilizando dois métodos presentes no QApedia, sendo eles o *QApedia.generator.perform_query()* e o *QApedia.generator.get_results_of_generator_query()*. O primeiro pode ser utilizado com qualquer consulta SPARQL, o segundo utiliza o primeiro método, mas antes ele ajusta a *generator_query* com a função explicada na seção *Ajustando a query geradora*. Como o resultado é grande, vamos apenas imprimir o tamanho da lista gerada e um exemplo.

```
>>> from QApedia import generator
>>> query = \"select distinct ?a ?b where {\\
...       ?uri <http://dbpedia.org/property/notableworks> ?a .\\
...       ?uri <http://dbpedia.org/property/notableworks> ?b .\\
...       ?uri a <http://dbpedia.org/ontology/Writer>}\"
>>> results = generator.perform_query(query)
>>> len(results)
10000
>>> results[15]
{'a': Value(typed-literal:'Petty Crimes'), 'b': Value(typed-literal:'New and Selected
↪Poems')}
```

3.5 Geração de pares

A principal funcionalidade do pacote reside na geração de pares de questão-sparql a partir de um template previamente estabelecido. Inicialmente, suponha que temos o seguinte template. Neste template desejamos construir uma lista de perguntas que contenha as obras do autor Yoshihiro Togashi.

Uma das perguntas possíveis seria: «Yoshihiro Togashi é autor de Hunter × Hunter»?

question	query	generator_query
Yoshihiro Togashi é autor de <A>?	ask where { <A> dbo:author dbr:Yoshihiro_Togashi }	select distinct ?a where { ?a dbo:author dbr:Yoshihiro_Togashi }

3.5.1 Realizando a consulta

Esse template é definido como um dicionário, onde cada chave corresponde ao nome da coluna, a chave *variables* corresponde as variáveis do tipo ?letra localizadas entre *select* e o *where* da *generator_query*.

```
>>> template = {"question": "Yoshihiro Togashi é autor de <A>?",
...             "query": "ask where { <A> dbo:author dbr:Yoshihiro_Togashi }",
...             "generator_query": "select distinct ?a where { ?a dbo:author_
↳dbr:Yoshihiro_Togashi }",
...             "variables": ["a"]}
>>> gquery = template["generator_query"]
>>> variables = template["variables"]
```

Dado esse template, inicialmente, iremos realizar o ajuste da *generator_query* adicionando o *label* do recurso armazenado na variável *a*. Para isso, realizamos a chamada da função de ajuste da *generator_query*:

```
>>> from QApedia.generator import adjust_generator_query
>>> gquery = adjust_generator_query(gquery, variables)
>>> print(gquery)
select distinct ?a ?la where {?a rdfs:label ?la. FILTER(lang(?la) = 'pt'). ?a_
↳dbo:author dbr:Yoshihiro_Togashi }
```

Em seguida, utilizamos a função de busca *perform_query* para realizar essa consulta. A consulta é realizada sobre a base da DBpedia, então não precisamos mudar valor padrão da função *endpoint="http://dbpedia.org/sparql"*.

```
>>> from QApedia.generator import perform_query
>>> results = perform_query(gquery)
>>> for instance in results:
...     print("%s: %s" %(instance["la"].value, instance["a"].value))
...
Level E: http://dbpedia.org/resource/Level_E
Yu Yu Hakusho: http://dbpedia.org/resource/Yu_Yu_Hakusho
Hunter × Hunter: http://dbpedia.org/resource/Hunter_×_Hunter
```

Outra forma de obter os resultados a partir da *generator_query* é utilizando o método *get_results_of_generator_query*.

```
>>> from QApedia.generator import get_results_of_generator_query
>>> generator_query = "select distinct ?a where { ?a dbo:author dbr:Yoshihiro_Togashi_
↳}"
>>> variables = ["a"]
>>> results = get_results_of_generator_query(generator_query, variables)
>>> for instance in results:
...     print("%s: %s" %(instance["la"].value, instance["a"].value))
...
Level E: http://dbpedia.org/resource/Level_E
Yu Yu Hakusho: http://dbpedia.org/resource/Yu_Yu_Hakusho
Hunter × Hunter: http://dbpedia.org/resource/Hunter_×_Hunter
```

3.5.2 Construindo os pares

Para a geração dos pares de questão-sparql, utilizamos a função *extract_pairs*, ela recebe como parâmetro:

- **resultado** da busca no formato retornado pelo exemplo anterior
- **template** da busca
- **quantidade de pares** que você deseja gerar, se a busca retornar mais do que esse valor, ela é limitada por essa quantidade.

- **lista de prefixos**, caso deseje que os recursos retornados no formato `<http://dbpedia.org/...>` sejam substituídos pelo prefixo especificado.

Exemplo 1

```
>>> from QApedia.generator import extract_pairs
>>> pairs = extract_pairs(results, template, 2)
>>> for pair in pairs:
...     print(pair["question"])
...     print(pair["sparql"])
...     print("----")
...
Yoshihiro Togashi é autor de Level E?
ask where { <http://dbpedia.org/resource/Level_E> dbo:author dbr:Yoshihiro_Togashi }
----
Yoshihiro Togashi é autor de Yu Yu Hakusho?
ask where { <http://dbpedia.org/resource/Yu_Yu_Hakusho> dbo:author dbr:Yoshihiro_
↪Togashi }
----
```

Exemplo 2

```
>>> from QApedia.generator import extract_pairs
>>> from QApedia.utils import convert_prefixes_to_list
>>> prefixes = "PREFIX dbr:<http://dbpedia.org/resource/>\n"
...             PREFIX dbo:<http://dbpedia.org/ontology/>"
>>> list_of_prefixes = convert_prefixes_to_list(prefixes)
>>> list_of_prefixes
[('dbr:', 'http://dbpedia.org/resource/'), ('dbo:', 'http://dbpedia.org/ontology/')]
>>> pairs = extract_pairs(results, template, 2, list_of_prefixes)
>>> for pair in pairs:
...     print(pair["question"])
...     print(pair["sparql"])
...     print("----")
...
Yoshihiro Togashi é autor de Level E?
ask where { dbr:Level_E dbo:author dbr:Yoshihiro_Togashi }
----
Yoshihiro Togashi é autor de Yu Yu Hakusho?
ask where { dbr:Yu_Yu_Hakusho dbo:author dbr:Yoshihiro_Togashi }
----
```

Caso deseje substituir alguns símbolos da sparql por elementos textuais, você pode fazer isso através da função `encode`. Para retornar a sparql em um formato válido, basta utilizar o método `decode`.

```
>>> from QApedia.utils import encode, decode
>>> for pair in pairs:
...     encoded = encode(pair["sparql"], list_of_prefixes)
...     decoded = decode(encoded, list_of_prefixes)
...     print(pair["question"])
...     print("====Encoded sparql====")
...     print(encoded)
...     print("====Decoded sparql====")
...     print(decoded)
...     print("----")
...
Yoshihiro Togashi é autor de Level E?
====Encoded sparql====
ask where bracket_open dbr_Level_E dbo_author dbr_Yoshihiro_Togashi bracket_close
```

(continues on next page)

(continuação da página anterior)

```
====Decoded sparql====
ask where { dbr:Level_E dbo:author dbr:Yoshihiro_Togashi }
----
Yoshihiro Togashi é autor de Yu Yu Hakusho?
====Encoded sparql====
ask where  bracket_open  dbr_Yu_Yu_Hakusho dbo_author dbr_Yoshihiro_Togashi  bracket_
↪close
====Decoded sparql====
ask where { dbr:Yu_Yu_Hakusho dbo:author dbr:Yoshihiro_Togashi }
----
```

4.1 Módulos

4.2 QApedia.qapedia module

Realiza a geração de pares questão-sparql a partir de um arquivo de templates previamente estabelecido.

4.2.1 Usage:

```
$ qapedia [-h] [-tfile TFILE] [-o OUTPUT] [-d DELIMITER] [-n NUMBER] [-p PREFIXES] [-e END-POINT] [-l LANG] [-v VERBOSE]
```

As opções disponíveis são:

-tfile	Qualquer caminho de string válido é aceito. A string pode ser uma URL, por exemplo. Esse caminho corresponde ao arquivo contendo o conjunto de templates. Se nenhum valor for passado, é executado um arquivo de exemplo.
-o, --output	Corresponde ao caminho do arquivo de saída onde será salvo os pares de questão-sparql gerados. Se nenhum caminho for especificado, o resultado será salvo no arquivo output.txt
-h, --help	Mostra informações sobre os argumentos.
-d, --delim	Delimitador usado para separar os campos do template. (default: “;”)
-n, --number	Quantidade de pares gerados por template. (default: 100)
-p, --prefixe	Caminho do arquivo txt contendo os prefixos utilizados, caso nenhum arquivo seja especificado são utilizados os mesmos prefixos presentes em http://dbpedia.org/snorql/
-e, --endpoint	URL do SPARQL endpoint. (default http://dbpedia.org/sparql)

-l, --lang Idioma das questões do template. (default: “pt”)

4.2.2 Examples:

```
$ qapedia -tfile templates.csv --lang PT
$ qapedia -h # exibe ajuda
```

4.2.3 Contact:

Mais informações estão disponíveis em:

- <https://qapedia.readthedocs.io/>
- <https://github.com/QApedia/QApedia>

4.2.4 Version:

- QApedia v0.0.0-alpha

4.3 QApedia.generator module

O módulo `generator` permite ao usuário realizar buscas sobre o endpoint da dbpedia. Além disso, permite ao usuário realizar a construção de queries sparql dado um template previamente especificado.

Este arquivo pode ser importado como um módulo e contém as seguintes funções:

- `adjust_generator_query` - retorna a `generator_query` com os rótulos correspondente a cada variável.
- `perform_query` - realiza a execução da query no endpoint da dbpedia.
- `get_results_of_generator_query` - similar a função `perform_query`, entretanto, realiza os ajustes em cima da `generator_query` e salva o resultado da busca na memória.
- `extract_pairs` - realiza a construção dos pares de questão-sparql com base no resultado e template especificados.
- `build_pairs_from_template` - realiza a construção de pares questão-sparql com base em um template previamente estabelecido.

`QApedia.generator.adjust_generator_query(generator_query, variables, lang='pt')`

Dada uma `generator_query` é retornada uma versão contendo os labels que são utilizados para preencher as lacunas presentes na pergunta.

Parameters

generator_query [str] Query utilizada para geração dos pares de questão-sparql.

variables [list] Lista contendo as variáveis utilizadas nas lacunas da questão-sparql.

lang [str, optional] Idioma do campo `rdfs:label` adicionado na `generator_query`. O valor padrão é “pt”.

Returns

str Retorna a `generator_query` com os campos `labels` de cada variável.

Examples

No exemplo a seguir, temos a `generator_query` que será utilizada futuramente para retornar recursos que tenham o campo `dbo:abstract`. O resultado dela é usado para preencher as lacunas do seguinte par ("o que é <A>?", "select ?a where { <A> dbo:abstract ?a "). Para preencher a lacuna da pergunta em linguagem natural, é adicionada na `generator_query` o campo `rdfs:label` correspondente as variáveis que se deseja obter informações.

```
>>> generator_query = "select distinct(?a) WHERE { ?a dbo:abstract []}"
>>> variables = ['a']
>>> result = adjust_generator_query(generator_query, variables)
>>> result
"select distinct(?a) ?la where { ?a rdfs:label ?la. FILTER(lang(?la) = 'pt'). ?a_
↳dbo:abstract [] }"
```

`QApedia.generator.perform_query(query, prefixes="", endpoint='http://dbpedia.org/sparql')`

Dada uma query sparql retorna uma lista correspondendo ao resultado da pesquisa se a cláusula utilizada for SELECT, CONSTRUCT ou DESCRIBE. Caso seja ASK, o valor retornado é um *boolean*.

Parameters

query [str] Sparql utilizada para realizar uma busca no endpoint especificado.

prefixes [str, optional] Corresponde ao conjunto de prefixos utilizados na consulta SPARQL. Se não estiver usando prefixos, o uso desse parâmetro não é necessário, o valor padrão é "".

endpoint [str, optional] Indica endpoint utilizado, o valor default é `http://dbpedia.org/sparql`

Returns

list of dict Corresponde a um lista contendo binds retornados pela busca Sparql. Se a cláusula utiliza SELECT ou CONSTRUCT.

bool Se a cláusula ASK for afirmativa retorna True, caso contrário False.

Raises

exc_type Caso haja um erro que não seja proveniente do problema de acesso ao endpoint, por exemplo, uma query em um formato inválido, uma exceção é gerada.

Examples

```
>>> from QApedia.generator import perform_query
>>> query = "SELECT * WHERE { "\
...         "?manga a dbo:Manga ." "\
...         "?manga rdfs:label ?nome ." "\
...         "?manga dbo:author dbr:Yoshihiro_Togashi ." "\
...         "FILTER(lang(?nome) = 'pt').}"
>>> results = perform_query(query)
>>> for result in results:
...     print("%s: %s" %(result["nome"].value, result["manga"].value))
...
Level E: http://dbpedia.org/resource/Level_E
Yu Yu Hakusho: http://dbpedia.org/resource/Yu_Yu_Hakusho
Hunter x Hunter: http://dbpedia.org/resource/Hunter_x_Hunter
```

```
QApedia.generator.get_results_of_generator_query(generator_query,          vari-
                                                ables,          prefixes=",",          end-
                                                point='http://dbpedia.org/sparql',
                                                lang='pt')
```

Dada uma `generator\_query` é retornado um conjunto de resultados obtidos ao executar a query no endpoint especificado.

Parameters

generator_query [str] String representando a `generator\_query`.

variables [list] Lista de caracteres correspondendo as variáveis.

prefixes: str, optional Corresponde ao conjunto de prefixos utilizados na consulta SPARQL. Se não estiver usando prefixos, o uso desse parâmetro não é necessário, o valor padrão é "".

endpoint [str, optional] Indica endpoint utilizado., by default "<http://dbpedia.org/sparql>"

lang [str, optional] Idioma do campo `rdfs:label` adicionado na `generator_query`. O valor padrão é "pt".

Returns

list of dict Corresponde a um lista contendo `bindings` retornados pela busca Sparql. Se a cláusula utiliza SELECT ou CONSTRUCT.

bool Se a cláusula ASK for afirmativa retorna True, caso contrário False.

```
QApedia.generator.extract_pairs(bindings,          template,          number_of_examples=500,
                                list_of_prefixes=[])
```

Realiza a extração do conjunto de pares de questão-sparql correspondentes obtidos pelo método `QApedia.generator.get_bindings_of_generator_query()`.

Parameters

bindings [list] Resultado obtido após a execução da query correspondendo aos «bindings»

template [dict] Corresponde ao template utilizado para geração dos resultados.

number_of_examples [int, optional] Número de resultados a serem considerados para o template, o valor padrão é 500.

list_of_prefixes [list, optional] Corresponde a lista de prefixos obtida através do método `QApedia.utils.convert_prefixes_to_list()`, onde os prefixos devem ser os mesmos que foram utilizados na função que gerou os bindings. Se não estiver usando prefixos, o uso desse parâmetro não é necessário, o valor padrão é [].

Returns

list Lista contendo os pares sparql-question do template.

Examples

```
>>> from QApedia.generator import extract_pairs
>>> from QApedia.generator import get_results_of_generator_query
>>> template = {"question": "Yoshihiro Togashi escreveu <A>?",
...             "query": "ask where {\n
...                 \"dbr:Yoshihiro_Togashi ^ dbo:author <A>\"},
...             \"generator_query\": \"select ?a where{\n
...                 \"dbr:Yoshihiro_Togashi ^ dbo:author ?a\"},
...             \"variables\": [\"a\"]}
>>> bindings = get_results_of_generator_query(
```

(continues on next page)

(continuação da página anterior)

```

...                                     template["generator_query"],
...                                     template["variables"])
>>> pairs = extract_pairs(bindings, template)
>>> pairs[2]["question"]
'Yoshihiro Togashi escreveu Hunter × Hunter?'
>>> pairs[2]["sparql"]
'ask where {dbr:Yoshihiro_Togashi ^ dbo:author http://dbpedia.org/resource/Hunter_
↳×_Hunter}'

```

`QApedia.generator.build_pairs_from_template` (*template*, *prefixes*=", *list_of_prefixes*=[], *endpoint*='http://dbpedia.org/sparql', *number_of_examples*=100, *lang*='pt')

Método responsável pela geração de pares questão-sparql com base em um template.

Parameters

template [dict] Representa um dicionário contendo os campos `question`, `query`, `generator_query` e `variables`.

prefixes [str, optional] Consiste em uma string contendo os prefixos utilizados pela SPARQL.

list_of_prefixes [list of tuple of str, optional, by default ""] Lista contendo os prefixos transformados pelo método `QApedia.utils.convert_prefixes_to_list()`, by default []

endpoint [str, optional] Corresponde ao SPARQL endpoint utilizado, by default «<http://dbpedia.org/sparql>»

number_of_examples [int, optional] Quantidade máxima de pares gerados por template, by default 100

lang [str, optional] Idioma utilizado na pergunta do template, by default "pt"

Returns

list of dict Lista contendo os pares de questão-sparql gerados para o template especificado.

Examples

```

>>> from QApedia.generator import build_pairs_from_template
>>> from QApedia.utils import convert_prefixes_to_list
>>> template = {"question": "Yoshihiro Togashi escreveu <A>?",
...             "query": "ask where {\\"
...                 \"dbr:Yoshihiro_Togashi ^ dbo:author <A>}\",
...             "generator_query": "select ?a where{\\"
...                 \"dbr:Yoshihiro_Togashi ^ dbo:author ?a}\",
...             "variables": ["a"]}
>>> prefixes = "PREFIX dbr: <http://dbpedia.org/resource/>\\"
...           "PREFIX dbo: <http://dbpedia.org/resource/>"
>>> list_of_prefixes = convert_prefixes_to_list(prefixes)
>>> pairs = build_pairs_from_template(template, prefixes,
...                                   list_of_prefixes)
>>> pairs[2]["question"]
'Yoshihiro Togashi escreveu Hunter × Hunter?'
>>> pairs[2]["sparql"]
'ask where {dbr:Yoshihiro_Togashi ^ dbo:author dbr:Hunter_×_Hunter}'

```

4.4 QApedia.io module

Este módulo trata das operações relacionadas a leitura e escrita do pacote QApedia.

Neste módulo, pode-se encontrar as seguintes funções:

- `load_templates` - realiza a leitura do arquivo contendo o conjunto de templates utilizados para a geração de perguntas-queries.
- `load_prefixes` - realiza a leitura do arquivo contendo os prefixos utilizados pelas SPARQLs durante a consulta.

`QApedia.io.load_templates(filepath, delimiter=';')`

A função `load_templates`, carrega o conjunto de templates a partir de um arquivo csv. O dado deve possuir um campo `generator_query` que servirá para realizar buscas que preencherão as lacunas presentes nos campos `question` e `query`.

Parameters

filepath [str] Caminho do arquivo csv que contém os templates.

delimiter [str, optional] Indicar qual separador utilizado no arquivo, by default “;”

Returns

pd.DataFrame Retorna um dataframe contendo o conjunto de templates.

Examples

Exemplo contendo 14 templates sendo carregado através da função `load_templates`.

```
>>> from QApedia.io import load_templates
>>> filename = "sample.csv"
>>> templates = load_templates(filename)
>>> len(templates)
14
>>> templates.head()
      query ... variables
0  <A> e <B> são os municípios vizinhos de que lu...  ...  [a, b]
1              <A> e <B> pertencem a qual espécie?  ...  [a, b]
2      <A> e <B> podem ser encontrados em qual país?  ...  [a, b]
3              <A> e <B> é produzido por qual empresa?  ...  [a, b]
4      <A> e <B> é o trabalho notável de qual autor?  ...  [a, b]

[5 rows x 4 columns]
```

`QApedia.io.load_prefixes(filepath)`

Dado um arquivo txt contendo os prefixos utilizados na SPARQL, é devolvida uma string contendo os prefixos e uma lista de tuplas contendo os prefixos.

Parameters

filepath [str] Caminho do arquivo txt contendo o conjunto de prefixos.

Returns

tuple of str Uma tupla contendo os prefixos carregados na forma de string e uma lista de tuplas, onde a primeira posição é o nome dado ao URI e a segunda contém a URI correspondente.

Examples

```
>>> from QApedia.io import load_prefixes
>>> filename = "prefixes.txt"
>>> prefixes = load_prefixes(filename)
>>> for uri_name, uri in prefixes[1]:
...     print(uri_name, uri)
...
owl: http://www.w3.org/2002/07/owl#
xsd: http://www.w3.org/2001/XMLSchema#
rdfs: http://www.w3.org/2000/01/rdf-schema#
rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
foaf: http://xmlns.com/foaf/0.1/
dc: http://purl.org/dc/elements/1.1/
dbpedia2: http://dbpedia.org/property/
dbpedia: http://dbpedia.org/
skos: http://www.w3.org/2004/02/skos/core#
```

4.5 QApedia.utils module

Este módulo contém o conjunto de operações utilizadas pelos módulos principais como por exemplo, o método `QApedia.io.load_templates()` que utiliza o método `QApedia.utils.extract_variables()` desse módulo para extrair o conjunto de variáveis presentes na query geradora (`generator_query`).

Neste módulo, pode-se encontrar as seguintes funções:

- `extract_variables` - realiza a extração das variáveis presentes no select da query geradora.
- `convert_prefixes_to_list` - dado o conjunto de prefixos, converte a string em uma lista de tuplas.
- `encode` - dada uma sparql realiza a codificação da query transformando alguns símbolos em texto.
- `decode` - dada uma sparql transformada por meio do `encode` realiza a transformação de inversa, de modo a substituir o texto por operações válidas.

`QApedia.utils.extract_variables(generator_query)`

Extrai as variáveis correspondente presentes no “generator_query”.

Parameters

generator_query [str] A “generator_query” corresponde a query que será utilizada para obter as variáveis presente nas lacunas da pergunta(query) e do sparql.

Returns

list Lista contendo as variáveis a serem respondidas.

Examples

```
>>> generator_query = "select distinct ?a where {"\
...     "?uri <http://dbpedia.org/ontology/author> ?a }"
>>> variables = extract_variables(generator_query)
>>> print(variables)
['a']
```

`QApedia.utils.convert_prefixes_to_list` (*prefixes*)

Converte uma string dos prefixos em uma lista de tuplas. Onde cada par contém um identificador e a uri correspondente.

Parameters

prefixes [str] string correspondendo aos prefixos utilizados na consulta SPARQL.

Returns

list Lista de tuplas, onde cada tupla contém dois itens, o primeiro corresponde ao nome dado a URI que corresponde ao segundo item.

Examples

```
>>> from QApedia.utils import convert_prefixes_to_list
>>> prefixes = "PREFIX foaf: <http://xmlns.com/foaf/0.1/>\
...           PREFIX yago: <http://yago-knowledge.org/resource/>\
...           PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>\
...           PREFIX dbo:<http://dbpedia.org/ontology/>\
...           PREFIX dbp:<http://dbpedia.org/property/>"
>>> list_of_prefixes = convert_prefixes_to_list(prefixes)
>>> for prefix, uri in list_of_prefixes:
...     print(prefix, uri)
...
foaf: http://xmlns.com/foaf/0.1/
yago: http://yago-knowledge.org/resource/
rdfs: http://www.w3.org/2000/01/rdf-schema#
dbo: http://dbpedia.org/ontology/
dbp: http://dbpedia.org/property/
```

`QApedia.utils.encode` (*sparql, prefixes*)

Dada uma query sparql, essa função transforma algum de seus caracteres em texto.

Parameters

sparql [str] sparql a ser transformada.

prefixes [list] lista de prefixos com uris utilizadas na sparql retornadas pela função `QApedia.utils.convert_prefixes_to_list()`.

Returns

str sparql transformada.

Examples

```
>>> from QApedia.utils import encode
>>> from QApedia.utils import convert_prefixes_to_list
>>> prefixes = "PREFIX prop: <http://dbpedia.org/property/>\
...           PREFIX dbr: <http://dbpedia.org/resource/>"
>>> query = "ASK {\n\
...         <http://dbpedia.org/resource/Amazon_River> prop:length      ?amazon .\
...         ↪\n\
...         <http://dbpedia.org/resource/Nile> prop:length ?nile .\n\
...         FILTER(?amazon > ?nile) .\n\
...         }"
>>> list_of_prefixes = convert_prefixes_to_list(prefixes)
```

(continues on next page)

(continuação da página anterior)

```

>>> query_encoded = encode(query, list_of_prefixes)
>>> print(query_encoded)
ASK  bracket_open
      dbr_Amazon_River prop_length      var_amazon  sep_dot
      dbr_Nile prop_length var_nile  sep_dot
      FILTER(var_amazon  greater_than  var_nile)  sep_dot
      bracket_close

```

`QApedia.utils.decode(sparql_encoded, prefixes)`

Dada uma sparql que foi codificada pela função `QApedia.utils.encode()`. O método `decode` substitui os termos codificados por símbolos válidos da consulta sparql.

Parameters

sparql_encoded [str] sparql transformada após passar por `QApedia.utils.encode()`.

prefixes [list] lista de prefixos com uris utilizadas na sparql retornadas pela função `QApedia.utils.convert_prefixes_to_list()`.

Returns

str sparql com os símbolos válidos para uma consulta.

Examples

```

>>> from QApedia.utils import decode
>>> from QApedia.utils import convert_prefixes_to_list
>>> prefixes = "PREFIX prop: <http://dbpedia.org/property/>\n\
...           PREFIX dbr: <http://dbpedia.org/resource/>"
>>> list_of_prefixes = convert_prefixes_to_list(prefixes)
>>> query_encoded = "ASK  bracket_open \n\
...      dbr_Amazon_River prop_length var_amazon  sep_dot \n\
...      dbr_Nile prop_length var_nile  sep_dot \n\
...      FILTER(var_amazon  greater_than  var_nile)  sep_dot  \n\
...      bracket_close "
>>> print(decode(query_encoded, list_of_prefixes))
ASK {
  dbr:Amazon_River prop:length ?amazon .
  dbr:Nile prop:length ?nile .
  FILTER(?amazon > ?nile) .
}

```


q

`QApedia.generator`, [18](#)

`QApedia.io`, [22](#)

`QApedia.qapedia`, [17](#)

`QApedia.utils`, [23](#)

A

`adjust_generator_query()` (no módulo *QApedia.generator*), 18

B

`build_pairs_from_template()` (no módulo *QApedia.generator*), 21

C

`convert_prefixes_to_list()` (no módulo *QApedia.utils*), 23

D

`decode()` (no módulo *QApedia.utils*), 25

E

`encode()` (no módulo *QApedia.utils*), 24

`extract_pairs()` (no módulo *QApedia.generator*), 20

`extract_variables()` (no módulo *QApedia.utils*), 23

G

`get_results_of_generator_query()` (no módulo *QApedia.generator*), 19

L

`load_prefixes()` (no módulo *QApedia.io*), 22

`load_templates()` (no módulo *QApedia.io*), 22

P

`perform_query()` (no módulo *QApedia.generator*), 19

Q

QApedia.generator (módulo), 18

QApedia.io (módulo), 22

QApedia.qapedia (módulo), 17

QApedia.utils (módulo), 23