

---

# **MyGene.py Documentation**

***Release v3.0.0***

**Chunlei Wu**

**Jul 11, 2017**



---

## Contents

---

<b>1</b>	<b>Requirements</b>	<b>3</b>
<b>2</b>	<b>Optional dependencies</b>	<b>5</b>
<b>3</b>	<b>Installation</b>	<b>7</b>
<b>4</b>	<b>Version history</b>	<b>9</b>
<b>5</b>	<b>Tutorial</b>	<b>11</b>
<b>6</b>	<b>API</b>	<b>13</b>
<b>7</b>	<b>Indices and tables</b>	<b>19</b>
	<b>Python Module Index</b>	<b>21</b>



[MyGene.Info](#) provides simple-to-use REST web services to query/retrieve gene annotation data. It's designed with simplicity and performance emphasized. *mygene*, is an easy-to-use Python wrapper to access [MyGene.Info](#) services.



# CHAPTER 1

---

## Requirements

---

python >=2.6 (including python3)

requests (install using “pip install requests”)





## CHAPTER 2

---

### Optional dependencies

---

`pandas` (install using “`pip install pandas`”) is required for returning a list of gene objects as `DataFrame`.



## CHAPTER 3

---

### Installation

---

**Option 1** pip install mygene

**Option 2** download/extract the source code and run:

```
python setup.py install
```

**Option 3** install the latest code directly from the repository:

```
pip install -e git+https://github.com/sulab/mygene.py#egg=mygene
```



## CHAPTER 4

---

### Version history

---

CHANGES.txt



## CHAPTER 5

---

### Tutorial

---

- ID mapping using mygene module in Python





`mygene.alwayslist (value)`

If input value is not a list/tuple type, return it as a single value list.

Example:

```
>>> x = 'abc'
>>> for xx in alwayslist(x):
...     print xx
>>> x = ['abc', 'def']
>>> for xx in alwayslist(x):
...     print xx
```

`class mygene.MyGeneInfo (url='http://mygene.info/v3')`

This is the client for MyGene.info web services. Example:

```
>>> mg = MyGeneInfo()
```

`metadata (verbose=True, **kwargs)`

Return a dictionary of MyGene.info metadata.

Example:

```
>>> metadata = mg.metadata
```

`set_caching (cache_db='mygene_cache', verbose=True, **kwargs)`

Installs a local cache for all requests. **cache\_db** is the path to the local sqlite cache database.

`stop_caching ()`

Stop caching.

`clear_cache ()`

Clear the globally installed cache.

`get_fields (search_term=None, verbose=True)`

Return all available fields that can be returned from MyGene.info services.

This is a wrapper for <http://mygene.info/metadata/fields>

**Parameters** `search_term` – an optional string to search (case insensitive) for matching field names. If not provided, all available fields will be returned.

Example:

```
>>> mv.get_fields()
>>> mv.get_fields("uniprot")
>>> mv.get_fields("refseq")
>>> mv.get_fields("kegg")
```

---

**Hint:** This is useful to find out the field names you need to pass to **fields** parameter of other methods.

---

**getgene** (*geneid*, *fields*=*'symbol, name, taxid, entrezgene'*, *\*\*kwargs*)

Return the gene object for the given geneid. This is a wrapper for GET query of “/gene/<geneid>” service.

**Parameters**

- **geneid** – entrez/ensembl gene id, entrez gene id can be either a string or integer
- **fields** – fields to return, a list or a comma-separated string. If **fields=“all”**, all available fields are returned
- **species** – optionally, you can pass comma-separated species names or taxonomy ids
- **email** – optionally, pass your email to help us to track usage
- **filter** – alias for **fields** parameter

**Returns** a gene object as a dictionary, or None if geneid is not valid.

**Ref** [http://mygene.info/doc/annotation\\_service.html](http://mygene.info/doc/annotation_service.html) for available fields, extra *kwargs* and more.

Example:

```
>>> mg.getgene(1017, email='abc@example.com')
>>> mg.getgene('1017', fields='symbol,name,entrezgene,refseq')
>>> mg.getgene('1017', fields='symbol,name,entrezgene,refseq.rna')
>>> mg.getgene('1017', fields=['symbol', 'name', 'pathway.kegg'])
>>> mg.getgene('ENSG00000123374', fields='all')
```

---

**Hint:** The supported field names passed to **fields** parameter can be found from any full gene object (when **fields=“all”**). Note that field name supports dot notation for nested data structure as well, e.g. you can pass “refseq.rna” or “pathway.kegg”.

---

**getgenes** (*geneids*, *fields*=*'symbol, name, taxid, entrezgene'*, *\*\*kwargs*)

Return the list of gene objects for the given list of geneids. This is a wrapper for POST query of “/gene” service.

**Parameters**

- **geneids** – a list/tuple/iterable or comma-separated entrez/ensembl gene ids
- **fields** – fields to return, a list or a comma-separated string. If **fields=“all”**, all available fields are returned
- **species** – optionally, you can pass comma-separated species names or taxonomy ids
- **email** – optionally, pass your email to help us to track usage
- **filter** – alias for **fields**

- **as\_dataframe** – if True, return object as DataFrame (requires Pandas).
- **df\_index** – if True (default), index returned DataFrame by ‘query’, otherwise, index by number. Only applicable if as\_dataframe=True.

**Returns** a list of gene objects or a pandas DataFrame object (when **as\_dataframe** is True)

**Ref** [http://mygene.info/doc/annotation\\_service.html](http://mygene.info/doc/annotation_service.html) for available fields, extra *kwargs* and more.

Example:

```
>>> mg.getgenes([1017, '1018', 'ENSG00000148795'], email='abc@example.com')
>>> mg.getgenes([1017, '1018', 'ENSG00000148795'], fields="entrezgene,uniprot")
>>> mg.getgenes([1017, '1018', 'ENSG00000148795'], fields="all")
>>> mg.getgenes([1017, '1018', 'ENSG00000148795'], as_dataframe=True)
```

**Hint:** A large list of more than 1000 input ids will be sent to the backend web service in batches (1000 at a time), and then the results will be concatenated together. So, from the user-end, it’s exactly the same as passing a shorter list. You don’t need to worry about saturating our backend servers.

**query** (*q*, *\*\*kwargs*)

Return the query result. This is a wrapper for GET query of “/query?q=<query>” service.

#### Parameters

- **q** – a query string, detailed query syntax [here](#)
- **fields** – fields to return, a list or a comma-separated string. If **fields=“all”**, all available fields are returned
- **species** – optionally, you can pass comma-separated species names or taxonomy ids. Default: human,mouse,rat.
- **size** – the maximum number of results to return (with a cap of 1000 at the moment). Default: 10.
- **skip** – the number of results to skip. Default: 0.
- **sort** – Prefix with “-” for descending order, otherwise in ascending order. Default: sort by matching scores in decending order.
- **entrezonly** – if True, return only matching entrez genes, otherwise, including matching Ensemble-only genes (those have no matching entrez genes).
- **email** – optionally, pass your email to help us to track usage
- **as\_dataframe** – if True, return object as DataFrame (requires Pandas).
- **df\_index** – if True (default), index returned DataFrame by ‘query’, otherwise, index by number. Only applicable if as\_dataframe=True.
- **fetch\_all** – if True, return a generator to all query results (unsorted). This can provide a very fast return of all hits from a large query. Server requests are done in blocks of 1000 and yielded individually. Each 1000 block of results must be yielded within 1 minute, otherwise the request will expire on the server side.

**Returns** a dictionary with returned gene hits or a pandas DataFrame object (when **as\_dataframe** is True)

**Ref** [http://mygene.info/doc/query\\_service.html](http://mygene.info/doc/query_service.html) for available fields, extra *kwargs* and more.

Example:

```
>>> mg.query('cdk2')
>>> mg.query('reporter:1000_at')
>>> mg.query('symbol:cdk2', species='human')
>>> mg.query('symbol:cdk*', species=10090, size=5, as_dataframe=True)
>>> mg.query('q=chrX:151073054-151383976', species=9606)
```

**querymany** (*qterms*, *scopes=None*, *\*\*kwargs*)

Return the batch query result. This is a wrapper for POST query of “/query” service.

#### Parameters

- **qterms** – a list/tuple/iterable of query terms, or a string of comma-separated query terms.
- **scopes** – type of types of identifiers, either a list or a comma-separated fields to specify type of input qterms, e.g. “entrezgene”, “entrezgene,symbol”, [“ensemblgene”, “symbol”]. Refer to [official MyGene.info docs](http://mygene.info/docs) for full list of fields.
- **fields** – fields to return, a list or a comma-separated string. If **fields=“all”**, all available fields are returned
- **species** – optionally, you can pass comma-separated species names or taxonomy ids. Default: human,mouse,rat.
- **entrezonly** – if True, return only matching entrez genes, otherwise, including matching Ensembl-only genes (those have no matching entrez genes).
- **returnall** – if True, return a dict of all related data, including dup. and missing qterms
- **verbose** – if True (default), print out information about dup and missing qterms
- **email** – optionally, pass your email to help us to track usage
- **as\_dataframe** – if True, return object as DataFrame (requires Pandas).
- **df\_index** – if True (default), index returned DataFrame by ‘query’, otherwise, index by number. Only applicable if as\_dataframe=True.

**Returns** a list of gene objects or a pandas DataFrame object (when **as\_dataframe** is True)

**Ref** [http://mygene.info/doc/query\\_service.html](http://mygene.info/doc/query_service.html) for available fields, extra *kwargs* and more.

Example:

```
>>> mg.querymany(['DDX26B', 'CCDC83'], scopes='symbol', species=9606)
>>> mg.querymany(['1255_g_at', '1294_at', '1316_at', '1320_at'], scopes=
↳ 'reporter')
>>> mg.querymany(['NM_003466', 'CDK2', 695, '1320_at', 'Q08345'],
...               scopes='refseq,symbol,entrezgene,reporter,uniprot', species=
↳ 'human')
>>> mg.querymany(['1255_g_at', '1294_at', '1316_at', '1320_at'], scopes=
↳ 'reporter',
...               fields='ensembl.gene,symbol', as_dataframe=True)
```

---

**Hint:** `querymany()` is perfect for doing id mappings.

---

---

**Hint:** Just like `getgenes()`, passing a large list of ids (>1000) to `querymany()` is perfectly fine.

---

**findgenes** (*id\_li*, *\*\*kwargs*)

Deprecated since version 2.0.0.

Use `querymany()` instead. It's kept here as an alias of `querymany()` method.



## CHAPTER 7

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





**m**

mygene, [13](#)



### A

`alwayslist()` (in module `mygene`), [13](#)

### C

`clear_cache()` (`mygene.MyGeneInfo` method), [13](#)

### F

`findgenes()` (`mygene.MyGeneInfo` method), [16](#)

### G

`get_fields()` (`mygene.MyGeneInfo` method), [13](#)

`getgene()` (`mygene.MyGeneInfo` method), [14](#)

`getgenes()` (`mygene.MyGeneInfo` method), [14](#)

### M

`metadata()` (`mygene.MyGeneInfo` method), [13](#)

`mygene` (module), [13](#)

`MyGeneInfo` (class in `mygene`), [13](#)

### Q

`query()` (`mygene.MyGeneInfo` method), [15](#)

`querymany()` (`mygene.MyGeneInfo` method), [16](#)

### S

`set_caching()` (`mygene.MyGeneInfo` method), [13](#)

`stop_caching()` (`mygene.MyGeneInfo` method), [13](#)