

---

# **Crawler Documentation**

***Release 1.0.2***

**Nathan Seva, Hugo Posnic**

**Apr 08, 2018**



---

## Contents

---

<b>1</b>	<b>Modules</b>	<b>3</b>
1.1	crawler main module . . . . .	3
1.2	statistiques module . . . . .	3
<b>2</b>	<b>Package</b>	<b>5</b>
2.1	swiftea_bot.module module . . . . .	5
2.2	swiftea_bot.data module . . . . .	6
2.3	swiftea_bot.file_manager module . . . . .	7
2.4	crawling.web_connection module . . . . .	8
2.5	crawling.connection module . . . . .	8
2.6	crawling.site_informations module . . . . .	9
2.7	crawling.searches module . . . . .	10
2.8	crawling.parsers module . . . . .	11
2.9	database.database module . . . . .	12
2.10	database.database_manager module . . . . .	13
2.11	database.database_swiftea module . . . . .	13
2.12	index.index module . . . . .	14
2.13	index.inverted_index module . . . . .	15
2.14	index.ftp_manager module . . . . .	16
2.15	index.ftp_swiftea module . . . . .	17
<b>3</b>	<b>Tests</b>	<b>19</b>
3.1	tests.run_tests . . . . .	19
3.2	tests.swiftea_bot_test . . . . .	19
3.3	tests.crawling_test . . . . .	20
3.4	tests.database_test . . . . .	21
3.5	tests.index_test . . . . .	21
3.6	tests.crawler_test . . . . .	22
3.7	tests.global_test . . . . .	22
3.8	tests.test_data . . . . .	22
<b>4</b>	<b>Indices and tables</b>	<b>23</b>
	<b>Python Module Index</b>	<b>25</b>



Crawler is an open source web crawler for Swiftea. It can't be run by contributors because it needs `private_data.py` which is not upload for obvious reasons.



Here are described the two executables of Swiftea-Crawler.

### 1.1 crawler main module

### 1.2 statistiques module

Display stats.

`stats.average(content)`

Calculate average.

**Parameters** `content` (*list*) – values

**Returns** average

`stats.compress_stats(filename)`

`stats.stats(dir_stats='data/stats/')`





These packages provide all functions and class that crawler need.

## 2.1 swiftea\_bot.module module

Define several functions for all crawler's class.

`swiftea_bot.module.can_add_doc(docs, new_doc)`

To avoid documents duplicate, look for all url doc.

Parse self.infos of Crawler and return True if new\_doc isn't in it.

### Parameters

- **docs** (*list*) – the documents to check
- **new\_doc** (*dict*) – the doc to add

**Returns** True if can add the doc

`swiftea_bot.module.convert_keys(inverted_index)`

Convert *str* words keys into *int* from inverted-index.

Json convert doc id key in str, must convert in int.

**Parameters** **inverted\_index** – inverted\_index to convert

**Typ** **inverted\_index** dict

**Returns** converted inverted-index

`swiftea_bot.module.create_dirs()`

Manage crawler's running.

**Test a lot of things:** create config directory

create doc file if doesn't exists

create config file if it doesn't exists

create links directory if it doesn't exists

create index directory if it doesn't exists

`swiftea_bot.module.def_links()`

Create directory of links if it doesn't exist

Ask to user what doing if there isn't basic links. Create a basic links file if user what it.

`swiftea_bot.module.errors(message, error_code)`

Write the error report with the time in errors file.

Normally call by `tell()` when a `error_code` parameter is given.

### Parameters

- **message** (*str*) – message to print and write
- **error\_code** (*int*) – error code

`swiftea_bot.module.is_index()`

Check if there is a saved inverted-index file.

**Returns** True if there is one

`swiftea_bot.module.remove_duplicates(old_list)`

Remove duplicates from a list.

**Parameters** **old\_list** (*list*) – list to clean

**Returns** list without duplicates

`swiftea_bot.module.stats_send_index(begining, end)`

Time spent between two sending of index

`swiftea_bot.module.stats_webpages(begining, end)`

Write the time in second to crawl 10 webpages.

### Parameters

- **begining** (*int*) – time before starting crawl 10 webpages
- **end** (*int*) – time after crawled 10 webpages

`swiftea_bot.module.tell(message, error_code="", severity=1)`

Manage newspaper.

Print in console what the program is doing and save this in a copy with time in an event file.

### Parameters

- **message** (*str*) – message to print and write
- **error\_code** (*int*) – (optional) error code, if given call `errors()` with given message
- **severity** (*int*) – 1 is default severity, -1 add 4 spaces befor message, 0 add 2 spaces befor the message, 2 uppercase and underline message.

## 2.2 swiftea\_bot.data module

Define required data by crawler.

## 2.3 swiftea\_bot.file\_manager module

Swiftea-Crawler use a lot of files. For example to config the app, save links... Here is a class that manage files of crawler.

**class** swiftea\_bot.file\_manager.FileManager

File manager for Swiftea-Crawler.

Save and read links, read and write configuration variables, read inverted-index from json saved file and from used file when sending it.

Create configuration file if it doesn't exists or read it.

**check\_size\_files** ()

**check\_stop\_crawling** ()

Check if the user wants to stop program.

**check\_size\_links** (*links*)

Check number of links in file.

**Parameters** *links* (*str*) – links saved in file

**get\_inverted\_index** ()

Get inverted-index in local.

Called after a connection error. Read a json file that contains the inverted-index. Delete this file after reading it.

**Returns** inverted-index

**get\_lists\_words** ()

Get lists words from data

Check for dirs lists words, create them if they don't exist.

**Returns** stopwords, badwords

**get\_url** ()

Get url of next webpage.

Check the size of curent reading links and increment it if over.

**Returns** url of webpage to crawl

**read\_inverted\_index** ()

Get inverted-index in local.

Called after sending inverted-index without error. Read all files created to send inverted-index.

**Returns** inverted-index

**save\_config** ()

Save all configurations in config file.

**save\_inverted\_index** (*inverted\_index*)

Save inverted-index in local.

Save it in a json file when we can't send it.

**Parameters** *inverted\_index* (*dict*) – inverted-index

**save\_links** (*links*)

Save found links in file.

Save links in a file without doublons.

**Parameters** `links` (*list*) – links to save

## 2.4 crawling.web\_connection module

Connection to webpage is managed by requests module. Those errors are waiting for: timeout with socket module and urllib3 module and all RequestException errors.

**class** `crawling.web_connection.WebConnection`

Manage the web connection with the page to crawl.

**check\_robots\_perm** (*url*)

Check robots.txt for permission.

**Parameters** `url` (*str*) – webpage url

**Returns** True if can crawl

**duplicate\_content** (*request1, url*)

Avoid param duplicate.

Compare source codes with params and whitout. Return url whitout params if it's the same content.

**Parameters** `request` (*requests.models.Response*) – request

**Returns** url, source code

**get\_code** (*url*)

Get source code of given url.

**Parameters** `url` (*str*) – url of webpage

**Returns** source code, True if no take links, score and new url (redirection)

**search\_encoding** (*headers, code*)

Search encoding of webpage in source code.

If an encoding is found in source code, score is 1, but if not score is 0 and encoding is utf-8.

**Parameters**

- **headers** (*dict*) – hearders of requests
- **code** (*str*) – source code

**Returns** encoding of webpage and it score

**send\_request** (*url*)

## 2.5 crawling.connection module

Define several functions WebConnection.

`crawling.connection.all_urls` (*request*)

Return all urls from request.history.

**Parameters**

- **request** (*requests.models.Response*) – request
- **first** (*str*) – list start with the url if given

**Returns** list of redirected urls, first is the last one

`crawling.connection.check_connection(url='https://github.com')`

Test internet connection.

Try to connect to a website.

**Parameters** `url` – url used to test the connection

**Returns** True if connected to internet

`crawling.connection.duplicate_content(code1, code2)`

Compare code1 and code2.

**Parameters**

- **code1** (*str*) – first code to compare
- **code2** (*str*) – second code to compare

`crawling.connection.is_nofollow(url)`

Check if take links.

Search !nofollow! at the end of url, remove it if found.

**Parameters** `url` (*str*) – webpage url

**Returns** True if nofollow and url

## 2.6 crawling.site\_informations module

After parsing source code, extracted data must be classified and cleaned. Here is a class that use the html parser and manage all results.

**class** `crawling.site_informations.SiteInformations`

Class to manage searches in source code.

**clean\_favicon** (*favicon, base\_url*)

Clean favicon.

**Parameters** `favicon` (*str*) – favicon url to clean

**Returns** cleaned favicon

**clean\_keywords** (*dirty\_keywords, language*)

Clean found keywords.

Delete stopwords, bad chars, two letter less word and split word1-word2

**Parameters** `keywords` (*list*) – keywords to clean

**Returns** list of cleaned keywords

**clean\_links** (*links, base\_url=None*)

Clean webpage's links: rebuild urls with base url and remove anchors, mailto, javascript, .index.

**Parameters** `links` (*list*) – links to clean

**Returns** cleaned links without duplicate

**detect\_language** (*keywords*)

Detect language of webpage if not given.

**Parameters** `keywords` (*list*) – keywords of webpage used for detecting

**Returns** language found

**get\_infos** (*url, code, nofollow, score*)

Manage all searches of webpage's informations.

**Parameters**

- **url** (*str*) – url of webpage
- **score** (*int*) – score of webpage
- **code** (*str*) – source code of webpage
- **nofollow** (*bool*) – if we take links of webpage

**Returns** links, title, description, key words, language, score, number of words

**sane\_search** (*keywords, language, max\_ratio=0.2*)

Filter pages not suitable for a young audience.

**Param** keywords: webpage's keywords

**Pram language** found website language

**Returns** True or False

**set\_listswords** (*stopwords, badwords*)

## 2.7 crawling.searches module

Define several functions SiteInformations.

**crawling.searches.capitalize** (*text*)

Upper the first letter of given text

**Parameters** **text** (*str*) – text

**Returns** text

**crawling.searches.clean\_link** (*url, base\_url=None*)

Clean a link.

Rebuild url with base url, pass mailto and javascript, remove anchors, pass if more than 5 queries, pass if more than 255 chars, remove /index.xxx, remove last /.

**Parameters**

- **url** (*str*) – links to clean
- **base\_url** – base url for rebuilding, can be None if

**Returns** cleaned link

**crawling.searches.clean\_text** (*text*)

Clean up text by removing tabulations, blanks and carriage returns.

**Parameters** **text** (*str*) – text to clean\_text

**Returns** cleaned text

**crawling.searches.get\_base\_url** (*url*)

Get base url using urlparse.

**Parameters** **url** (*str*) – url

**Returns** base url of given url

`crawling.searches.is_homepage(url)`

Check if url is the homepage.

If there is only two '/' and two '.' if www and one otherwise.

**Parameters** `url` (*str*) – url to check

**Returns** True or False

`crawling.searches.stats_links(stats)`

Write the number of links for statistics.

**Parameters** `stat` (*int*) – number of list in a webpage

## 2.8 crawling.parsers module

Data of webpage is provided by the python html.parser. There are two parsers: the first one for all informations and the second one only for encoding.

**class** `crawling.parsers.ExtractData`

Bases: `html.parser.HTMLParser`

Html parser to extract data.

`self.object`: the type of text for title, description and keywords

`dict(attrs).get('content')`: convert attrs in a dict and return the value

**Data that could be extracted:** title

language

description

links with nofollow and noindex

stylesheet

favicon

keywords: h1, h2, h3, strong, em

**handle\_charref** (*name*)

**handle\_data** (*data*)

Called when parser meet data.

**Parameters** `tag` (*str*) – starting tag

**handle\_endtag** (*tag*)

Called when parser meet an ending tag.

**Parameters**

- `tag` (*str*) – starting tag
- `attrs` (*list*) – attributes

**handle\_entityref** (*name*)

**handle\_starttag** (*tag, attrs*)

Called when parser meet a starting tag.

**Parameters**

- `tag` (*str*) – starting tag

- **attrs** (*list*) – attributes: [(‘name’, ‘language’), (‘content’, ‘fr’)]

**re\_init()**

Called when we meet html tag, put back all variables to default.

**class** `crawling.parsers.ExtractEncoding`

Bases: `html.parser.HTMLParser`

Html parser to extract encoding from source code.

**handle\_starttag** (*tag*, *attrs*)

Called when parser meet a starting tag.

**Parameters**

- **tag** (*str*) – starting tag
- **attrs** (*list*) – attributes

`crawling.parsers.can_append` (*url*, *rel*)

Check rel attrs to know if crawler can crawl the link.

Add !nofollow! at the end of the url if it can’t follow links of url.

**Parameters**

- **url** (*str*) – url to add
- **rel** (*str*) – rel attrs in a tag

**Returns** None if it can’t add it, otherwise return url

`crawling.parsers.meta` (*attrs*)

Manage searches in tags.

**We can find:** <meta name=’description’ content=’my description’/>

<meta name=’language’ content=’en’/>

<meta http-equiv=’content-language’ content=’en’/>

**Apram attrs** attributes of meta tag

**Returns** language, description, object

## 2.9 database.database module

Define several functions for DatabaseSwifta.

`database.database.convert_secure` (*url*)

Convert https to http and http to https.

**Parameters** **url** (*str*) – url to convert

**Returns** converted url

`database.database.url_is_secure` (*url*)

Check if given url is secure (https).

**Parameters** **url** (*str*) – url to check

**Returns** True if url is secure



## 2.10 database.database\_manager module

**class** database.database\_manager.DatabaseManager (*host, user, password, name*)

Class to manage queries to the database using PyMySQL.

How to: create a subclass

result, response = self.send\_command(command, data=tuple(), all=False)

if 'error' in response:

    print('An error occurred.')

where result are data asked and response a message.

### Parameters

- **host** (*str*) – hostname of the db server
- **user** (*str*) – username to use for connection
- **password** (*str*) – password to use for connection
- **name** (*str*) – name of database

**close\_connection** ()

Close database connection.

**connection** ()

Connect to database.

**send\_command** (*command, data=(), fetchall=False*)

Send a query to database.

Catch timeout and OperationalError.

### Parameters

- **data** (*tuple*) – data attached to query
- **fetchall** (*bool*) – True if return all results

**Returns** result of the query and status message

**set\_name** (*name*)

Set base name

**Parameters** **name** (*str*) – new base name

## 2.11 database.database\_swiftea module

**class** database.database\_swiftea.DatabaseSwiftea (*host, user, password, name, table*)

Bases: *database.database\_manager.DatabaseManager*

Class to manage Swiftea database.

### Parameters

- **host** (*str*) – hostname of the db server
- **user** (*str*) – username to use for connection
- **password** (*str*) – password to use for connection
- **name** (*str*) – name of database

**del\_one\_doc** (*url*, *table=None*)

Delete document corresponding to url.

**Parameters** **url** (*str*) – url of webpage

**Returns** status message

**doc\_exists** (*url*)

Check if *url* is in database.

**Parameters** **url** (*str*) – url corresponding to doc

**Returns** True if doc exists

**get\_doc\_id** (*url*)

Get id of a document in database.

**Parameters** **url** (*str*) – url of webpage

**Returns** id of webpage or None if not found

**https\_duplicate** (*old\_url*)

Avoid https and http duplicate.

If old url is secure (https), must delete insecure url if exists, then return secure url (old url). If old url is insecure (http), must delete it if secure url exists, then return secure url (new url)

**Parameters** **old\_url** (*str*) – old url

**Returns** url to add and url to delete

**insert** (*infos*)

Insert a new document in database.

**Parameters** **infos** (*dict* ()) – doc infos

**Returns** True is an error occurred

**send\_doc** (*webpage\_infos*)

Send document informations to database.

**Parameters** **infos** (*list*) – informations to send to database

**Returns** True if an error occurred

**suggestions** ()

Get the five first URLs from Suggestion table and delete them.

**Returns** list of url in Suggestion table and delete them

**update** (*infos*, *popularity*)

Update a document in database.

**Parameters**

- **infos** (*dict* ()) – doc infos
- **popularity** (*int*) – new doc popularity

**Returns** True is an error occurred

## 2.12 index.index module

Define several functions for inverted-index.

`index.index.count_files_index(index)`

Return number of file to download are uplaod

Parse languages and letters from the given index.

**Returns** int

`index.index.stats_dl_index(begining, end)`

Write the time to download inverted-index.

**Parameters**

- **begining** (*int*) – time download inverted-index
- **end** (*int*) – time after download inverted-index

`index.index.stats_ul_index(begining, end)`

Write the time to upload inverted-index.

**Parameters**

- **begining** (*int*) – time before send inverted-index
- **end** (*int*) – time after send inverted-index

## 2.13 index.inverted\_index module

**class** `index.inverted_index.InvertedIndex`

Manage inverted-index for crawler.

Inverted-index is a dict, each keys are language

-> values are a dict, each keys are first letter

-> values are dict, each keys are two first letters

-> values are dict, each keys are word

-> values are dict, each keys are id

-> values are int: tf

example: ['FR']['A']['av']['avion'][21] is tf of word 'avion' in doc 21 in french.

**add\_doc** (*keywords, doc\_id, language*)

Add all words of a doc in inverted-index.

**Parameters**

- **keywords** (*list*) – all word in doc\_id
- **doc\_id** (*int*) – id of the doc in database
- **language** (*str*) – language of word

**add\_word** (*word\_infos, doc\_id, nb\_words*)

Add a word in inverted-index.

**Parameters**

- **word\_infos** (*dict*) – word infos: word, language, occurence, first letter and two first letters
- **doc\_id** (*int*) – id of the doc in database
- **nb\_words** (*int*) – number of words in the doc\_id

**delete\_doc\_id** (*doc\_id*)

Delete a id in inverted-index.

**Parameters** **doc\_id** (*int*) – id to delete

**delete\_id\_word** (*word\_infos*, *doc\_id*)

Delete a id of a word in inverted-index

This method delete a word from a document. Remove a words from a doc.

**Parameters**

- **word\_infos** (*dict*) – word infos: word, language, first letter and two first letters
- **doc\_id** (*int*) – id of the doc in database

**delete\_word** (*word*, *language*, *first\_letter*, *filename*)

Delete a word in inverted-index.

**Parameters**

- **word** (*str*) – word to delete
- **language** (*str*) – language of word
- **first\_letter** (*str*) – first letter of word
- **filename** (*str*) – two first letters of word

**getInvertedIndex** ()

**Returns** inverted-index

**setInvertedIndex** (*inverted\_index*)

Define inverted-index at the beginning.

**Parameters** **inverted\_index** (*dict*) – inverted-index

## 2.14 index.ftp\_manager module

**class** `index.ftp_manager.FTPManager` (*host*, *user*="", *password*="", *port*=21)

Bases: `ftplib.FTP`

Class to connect to a ftp server more easily.

**Parameters**

- **host** (*str*) – hostname of the ftp server
- **user** (*str*) – username to use for connection
- **password** (*str*) – password to use for connection

**cd** (*path*)

Set the current directory on the server.

**Parameters** **path** (*str*) – path to set

**Returns** sever response

**connection** ()

Connect to ftp server.

Catch all\_errors of ftplib. Use utf-8 encoding.

**Returns** server welcome message

**countfiles** (*path*='.')

Count the file in the given path

**Parameters** **path** (*str*) – path to count

**Returns** number of files

**disconnect** ()

Quit connection to ftp server.

Close it if an error occurred while trying to quit it.

**Returns** server goodbye message or error message

**get** (*local\_filename*, *server\_filename*)

Download a file from ftp server.

It creates the file to download.

**Parameters**

- **local\_filename** (*str*) – local filename to create
- **server\_filename** (*str*) – server filename to download

**Returns** server response message or error message

**infos\_listdir** (*path*='.', *facts*=[])

Return the result of mlsd command of ftplib or a list whose first element is the error response.

**listdir** ()

Return the result of LIST command or a list whose first element is the error response.

**mkdir** (*dirname*)

Create a directory on the server.

**Parameters** **dirname** (*str*) – the directory path and name

**Returns** server response

**put** (*local\_filename*, *server\_filename*)

Upload a file into ftp server.

The file to upload must exist.

**Parameters**

- **local\_filename** (*str*) – local filename to upload
- **server\_filename** (*str*) – server filename to upload

**Returns** response of server

**exception** `index.ftp_manager.MyFtpError` (*value*)

Bases: `Exception`

How to use it: `raise MyFtpError('Error message')`

## 2.15 index.ftp\_swiftea module



Tests for Swiftea-Crawler using pytest.

### 3.1 tests.run\_tests

### 3.2 tests.swiftea\_bot\_test

```
class tests.swiftea_bot_test.SwifteaBotBaseTest
    Bases: object
        setup_method(_)

class tests.swiftea_bot_test.TestFileManager
    Bases: tests.swiftea_bot_test.SwifteaBotBaseTest
        test_check_stop_crawling()
        test_ckeck_size_links()
        test_get_inverted_index()
        test_get_lists_words()
        test_get_url()
        test_init()
        test_read_inverted_index()
        test_save_config()
        test_save_inverted_index()
        test_save_links()

class tests.swiftea_bot_test.TestModule
    Bases: tests.swiftea_bot_test.SwifteaBotBaseTest
```

```
test_can_add_doc()
test_create_dirs()
test_is_index()
test_remove_duplicates()
test_stats_webpages()
test_tell()
```

### 3.3 tests.crawling\_test

```
class tests.crawling_test.CrawlingBaseTest
    Bases: object

    Base class for all crawler test classes.

    setup_method(_)
        Configure the app.

class tests.crawling_test.TestConnection
    Bases: tests.crawling_test.CrawlingBaseTest

    test_all_urls()

    test_check_connection()

    test_duplicate_content()

    test_is_nofollow()

class tests.crawling_test.TestParsers
    Bases: tests.crawling_test.CrawlingBaseTest

    test_can_append()

    test_handle_charref()

    test_handle_entityref()

    test_meta()

    test_parser()

    test_parser_encoding()

class tests.crawling_test.TestSearches
    Bases: tests.crawling_test.CrawlingBaseTest

    test_capitalize()

    test_clean_link()

    test_clean_text()

    test_get_base_url()

    test_is_homepage()

    test_stats_links()

class tests.crawling_test.TestSiteInformations
    Bases: tests.crawling_test.CrawlingBaseTest
```



```

    test_clean_favicon()
    test_clean_keywords()
    test_clean_links()
    test_detect_language()
    test_sane_search()
    test_set_listswords()
class tests.crawling_test.TestWebConnection
    Bases: tests.crawling_test.CrawlingBaseTest
    test_check_robots_perm()
    test_duplicate_content()
    test_search_encoding()
    test_send_request()

```

### 3.4 tests.database\_test

```

class tests.database_test.DatabaseBaseTest
    Bases: object
    Base class for all crawler test classes.
    setup_method(_)
class tests.database_test.TestDatabase
    Bases: tests.database_test.DatabaseBaseTest
    test_convert_secure()
    test_url_is_secure()

```

### 3.5 tests.index\_test

```

class tests.index_test.IndexBaseTest
    Bases: object
    setup_method(_)
class tests.index_test.TestIndex
    Bases: tests.index_test.IndexBaseTest
    test_count_files_index()
    test_stats_dl_index()
    test_stats_ul_index()
class tests.index_test.TestInvertedIndex
    Bases: tests.index_test.IndexBaseTest
    test_add_word()
    test_create_inverted_index()
    test_delete_doc_id()

```

```
test_delete_id_word()  
test_delete_word()  
test_getInvertedIndex()  
test_setInvertedIndex()
```

### 3.6 tests.crawler\_test

### 3.7 tests.global\_test

### 3.8 tests.test\_data

```
tests.test_data.reset()
```

## CHAPTER 4

---

### Indices and tables

---

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



### c

`crawling.connection`, 8  
`crawling.parsers`, 11  
`crawling.searches`, 10  
`crawling.site_informations`, 9  
`crawling.web_connection`, 8

### d

`database.database`, 12  
`database.database_manager`, 13  
`database.database_swiftea`, 13

### i

`index.ftp_manager`, 16  
`index.index`, 14  
`index.inverted_index`, 15

### s

`stats`, 3  
`swiftea_bot.data`, 6  
`swiftea_bot.file_manager`, 7  
`swiftea_bot.module`, 5

### t

`tests.crawling_test`, 20  
`tests.database_test`, 21  
`tests.index_test`, 21  
`tests.swiftea_bot_test`, 19  
`tests.test_data`, 22



## A

add\_doc() (index.inverted\_index.InvertedIndex method), 15  
 add\_word() (index.inverted\_index.InvertedIndex method), 15  
 all\_urls() (in module crawling.connection), 8  
 average() (in module stats), 3

## C

can\_add\_doc() (in module swiftea\_bot.module), 5  
 can\_append() (in module crawling.parsers), 12  
 capitalize() (in module crawling.searches), 10  
 cd() (index.ftp\_manager.FTPManager method), 16  
 check\_connection() (in module crawling.connection), 8  
 check\_robots\_perm() (crawling.web\_connection.WebConnection method), 8  
 check\_size\_files() (swiftea\_bot.file\_manager.FileManager method), 7  
 check\_stop\_crawling() (swiftea\_bot.file\_manager.FileManager method), 7  
 check\_size\_links() (swiftea\_bot.file\_manager.FileManager method), 7  
 clean\_favicon() (crawling.site\_informations.SiteInformations method), 9  
 clean\_keywords() (crawling.site\_informations.SiteInformations method), 9  
 clean\_link() (in module crawling.searches), 10  
 clean\_links() (crawling.site\_informations.SiteInformations method), 9  
 clean\_text() (in module crawling.searches), 10  
 close\_connection() (database.database\_manager.DatabaseManager method), 13  
 compress\_stats() (in module stats), 3  
 connection() (database.database\_manager.DatabaseManager method), 13  
 connection() (index.ftp\_manager.FTPManager method),

16

convert\_keys() (in module swiftea\_bot.module), 5  
 convert\_secure() (in module database.database), 12  
 count\_files\_index() (in module index.index), 14  
 countfiles() (index.ftp\_manager.FTPManager method), 16  
 crawling.connection (module), 8  
 crawling.parsers (module), 11  
 crawling.searches (module), 10  
 crawling.site\_informations (module), 9  
 crawling.web\_connection (module), 8  
 CrawlingBaseTest (class in tests.crawling\_test), 20  
 create\_dirs() (in module swiftea\_bot.module), 5

## D

database.database (module), 12  
 database.database\_manager (module), 13  
 database.database\_swiftea (module), 13  
 DatabaseBaseTest (class in tests.database\_test), 21  
 DatabaseManager (class in database.database\_manager), 13  
 DatabaseSwiftea (class in database.database\_swiftea), 13  
 def\_links() (in module swiftea\_bot.module), 6  
 del\_one\_doc() (database.database\_swiftea.DatabaseSwiftea method), 14  
 delete\_doc\_id() (index.inverted\_index.InvertedIndex method), 15  
 delete\_id\_word() (index.inverted\_index.InvertedIndex method), 16  
 delete\_word() (index.inverted\_index.InvertedIndex method), 16  
 detect\_language() (crawling.site\_informations.SiteInformations method), 9  
 disconnect() (index.ftp\_manager.FTPManager method), 17  
 doc\_exists() (database.database\_swiftea.DatabaseSwiftea method), 14  
 duplicate\_content() (crawling.web\_connection.WebConnection method),

8  
duplicate\_content() (in module crawling.connection), 9

## E

errors() (in module swiftea\_bot.module), 6  
ExtractData (class in crawling.parsers), 11  
ExtractEncoding (class in crawling.parsers), 12

## F

FileManager (class in swiftea\_bot.file\_manager), 7  
FTPManager (class in index.ftp\_manager), 16

## G

get() (index.ftp\_manager.FTPManager method), 17  
get\_base\_url() (in module crawling.searches), 10  
get\_code() (crawling.web\_connection.WebConnection method), 8  
get\_doc\_id() (database.database\_swiftea.DatabaseSwiftea method), 14  
get\_infos() (crawling.site\_informations.SiteInformations method), 9  
get\_inverted\_index() (swiftea\_bot.file\_manager.FileManager method), 7  
get\_lists\_words() (swiftea\_bot.file\_manager.FileManager method), 7  
get\_url() (swiftea\_bot.file\_manager.FileManager method), 7  
getInvertedIndex() (index.inverted\_index.InvertedIndex method), 16

## H

handle\_charref() (crawling.parsers.ExtractData method), 11  
handle\_data() (crawling.parsers.ExtractData method), 11  
handle\_endtag() (crawling.parsers.ExtractData method), 11  
handle\_entityref() (crawling.parsers.ExtractData method), 11  
handle\_starttag() (crawling.parsers.ExtractData method), 11  
handle\_starttag() (crawling.parsers.ExtractEncoding method), 12  
https\_duplicate() (database.database\_swiftea.DatabaseSwiftea method), 14

## I

index.ftp\_manager (module), 16  
index.index (module), 14  
index.inverted\_index (module), 15  
IndexBaseTest (class in tests.index\_test), 21  
infos\_listdir() (index.ftp\_manager.FTPManager method), 17  
insert() (database.database\_swiftea.DatabaseSwiftea method), 14

InvertedIndex (class in index.inverted\_index), 15  
is\_homepage() (in module crawling.searches), 10  
is\_index() (in module swiftea\_bot.module), 6  
is\_nofollow() (in module crawling.connection), 9

## L

listdir() (index.ftp\_manager.FTPManager method), 17

## M

meta() (in module crawling.parsers), 12  
mkdir() (index.ftp\_manager.FTPManager method), 17  
MyFtpError, 17

## P

put() (index.ftp\_manager.FTPManager method), 17

## R

re\_init() (crawling.parsers.ExtractData method), 12  
read\_inverted\_index() (swiftea\_bot.file\_manager.FileManager method), 7  
remove\_duplicates() (in module swiftea\_bot.module), 6  
reset() (in module tests.test\_data), 22

## S

sane\_search() (crawling.site\_informations.SiteInformations method), 10  
save\_config() (swiftea\_bot.file\_manager.FileManager method), 7  
save\_inverted\_index() (swiftea\_bot.file\_manager.FileManager method), 7  
save\_links() (swiftea\_bot.file\_manager.FileManager method), 7  
search\_encoding() (crawling.web\_connection.WebConnection method), 8  
send\_command() (database.database\_manager.DatabaseManager method), 13  
send\_doc() (database.database\_swiftea.DatabaseSwiftea method), 14  
send\_request() (crawling.web\_connection.WebConnection method), 8  
set\_listswords() (crawling.site\_informations.SiteInformations method), 10  
set\_name() (database.database\_manager.DatabaseManager method), 13  
setInvertedIndex() (index.inverted\_index.InvertedIndex method), 16  
setup\_method() (tests.crawling\_test.CrawlingBaseTest method), 20  
setup\_method() (tests.database\_test.DatabaseBaseTest method), 21  
setup\_method() (tests.index\_test.IndexBaseTest method), 21



setup\_method() (tests.swiftea\_bot\_test.SwifteaBotBaseTest test\_create\_dirs() (tests.swiftea\_bot\_test.TestModule method), 19  
 SiteInformations (class in crawling.site\_informations), 9  
 stats (module), 3  
 stats() (in module stats), 3  
 stats\_dl\_index() (in module index.index), 15  
 stats\_links() (in module crawling.searches), 11  
 stats\_send\_index() (in module swiftea\_bot.module), 6  
 stats\_ul\_index() (in module index.index), 15  
 stats\_webpages() (in module swiftea\_bot.module), 6  
 suggestions() (database.database\_swiftea.DatabaseSwiftea method), 14  
 swiftea\_bot.data (module), 6  
 swiftea\_bot.file\_manager (module), 7  
 swiftea\_bot.module (module), 5  
 SwifteaBotBaseTest (class in tests.swiftea\_bot\_test), 19

## T

tell() (in module swiftea\_bot.module), 6  
 test\_add\_word() (tests.index\_test.TestInvertedIndex method), 21  
 test\_all\_urls() (tests.crawling\_test.TestConnection method), 20  
 test\_can\_add\_doc() (tests.swiftea\_bot\_test.TestModule method), 19  
 test\_can\_append() (tests.crawling\_test.TestParsers method), 20  
 test\_capitalize() (tests.crawling\_test.TestSearches method), 20  
 test\_check\_connection() (tests.crawling\_test.TestConnection method), 20  
 test\_check\_robots\_perm() (tests.crawling\_test.TestWebConnection method), 21  
 test\_check\_stop\_crawling() (tests.swiftea\_bot\_test.TestFileManager method), 19  
 test\_ckeck\_size\_links() (tests.swiftea\_bot\_test.TestFileManager method), 19  
 test\_clean\_favicon() (tests.crawling\_test.TestSiteInformations method), 20  
 test\_clean\_keywords() (tests.crawling\_test.TestSiteInformations method), 21  
 test\_clean\_link() (tests.crawling\_test.TestSearches method), 20  
 test\_clean\_links() (tests.crawling\_test.TestSiteInformations method), 21  
 test\_clean\_text() (tests.crawling\_test.TestSearches method), 20  
 test\_convert\_secure() (tests.database\_test.TestDatabase method), 21  
 test\_count\_files\_index() (tests.index\_test.TestIndex method), 21  
 test\_create\_dirs() (tests.swiftea\_bot\_test.TestModule method), 20  
 test\_create\_inverted\_index() (tests.index\_test.TestInvertedIndex method), 21  
 test\_delete\_doc\_id() (tests.index\_test.TestInvertedIndex method), 21  
 test\_delete\_id\_word() (tests.index\_test.TestInvertedIndex method), 21  
 test\_delete\_word() (tests.index\_test.TestInvertedIndex method), 22  
 test\_detect\_language() (tests.crawling\_test.TestSiteInformations method), 21  
 test\_duplicate\_content() (tests.crawling\_test.TestConnection method), 20  
 test\_duplicate\_content() (tests.crawling\_test.TestWebConnection method), 21  
 test\_get\_base\_url() (tests.crawling\_test.TestSearches method), 20  
 test\_get\_inverted\_index() (tests.swiftea\_bot\_test.TestFileManager method), 19  
 test\_get\_lists\_words() (tests.swiftea\_bot\_test.TestFileManager method), 19  
 test\_get\_url() (tests.swiftea\_bot\_test.TestFileManager method), 19  
 test\_getInvertedIndex() (tests.index\_test.TestInvertedIndex method), 22  
 test\_handle\_charref() (tests.crawling\_test.TestParsers method), 20  
 test\_handle\_entityref() (tests.crawling\_test.TestParsers method), 20  
 test\_init() (tests.swiftea\_bot\_test.TestFileManager method), 19  
 test\_is\_homepage() (tests.crawling\_test.TestSearches method), 20  
 test\_is\_index() (tests.swiftea\_bot\_test.TestModule method), 20  
 test\_is\_nofollow() (tests.crawling\_test.TestConnection method), 20  
 test\_is\_meta() (tests.crawling\_test.TestParsers method), 20  
 test\_parser() (tests.crawling\_test.TestParsers method), 20  
 test\_parser\_encoding() (tests.crawling\_test.TestParsers method), 20  
 test\_read\_inverted\_index() (tests.swiftea\_bot\_test.TestFileManager method), 19  
 test\_remove\_duplicates() (tests.swiftea\_bot\_test.TestModule method), 20  
 test\_sane\_search() (tests.crawling\_test.TestSiteInformations method), 21  
 test\_save\_config() (tests.swiftea\_bot\_test.TestFileManager method), 19  
 test\_save\_inverted\_index() (tests.swiftea\_bot\_test.TestFileManager method), 19

- method), 19
- test\_save\_links() (tests.swiftea\_bot\_test.TestFileManager method), 19
- test\_search\_encoding() (tests.crawling\_test.TestWebConnection method), 21
- test\_send\_request() (tests.crawling\_test.TestWebConnection method), 21
- test\_set\_listswords() (tests.crawling\_test.TestSiteInformations method), 21
- test\_setInvertedIndex() (tests.index\_test.TestInvertedIndex method), 22
- test\_stats\_dl\_index() (tests.index\_test.TestIndex method), 21
- test\_stats\_links() (tests.crawling\_test.TestSearches method), 20
- test\_stats\_ul\_index() (tests.index\_test.TestIndex method), 21
- test\_stats\_webpages() (tests.swiftea\_bot\_test.TestModule method), 20
- test\_tell() (tests.swiftea\_bot\_test.TestModule method), 20
- test\_url\_is\_secure() (tests.database\_test.TestDatabase method), 21
- TestConnection (class in tests.crawling\_test), 20
- TestDatabase (class in tests.database\_test), 21
- TestFileManager (class in tests.swiftea\_bot\_test), 19
- TestIndex (class in tests.index\_test), 21
- TestInvertedIndex (class in tests.index\_test), 21
- TestModule (class in tests.swiftea\_bot\_test), 19
- TestParsers (class in tests.crawling\_test), 20
- tests.crawling\_test (module), 20
- tests.database\_test (module), 21
- tests.index\_test (module), 21
- tests.swiftea\_bot\_test (module), 19
- tests.test\_data (module), 22
- TestSearches (class in tests.crawling\_test), 20
- TestSiteInformations (class in tests.crawling\_test), 20
- TestWebConnection (class in tests.crawling\_test), 21

## U

- update() (database.database\_swiftea.DatabaseSwiftea method), 14
- url\_is\_secure() (in module database.database), 12

## W

- WebConnection (class in crawling.web\_connection), 8