

---

# **Linguistic Analyzer Documentation**

***Release 2.0***

**Paul Brown, Tyler Blanton**

**Mar 19, 2018**



---

## Contents:

---

<b>1</b>	<b>Keyword module</b>	<b>1</b>
<b>2</b>	<b>KeywordList module</b>	<b>3</b>
<b>3</b>	<b>functionsv1 package</b>	<b>5</b>
3.1	common_functions module . . . . .	5
3.2	analyze_functions module . . . . .	9
<b>4</b>	<b>analyze module</b>	<b>13</b>
<b>5</b>	<b>application module</b>	<b>15</b>
<b>6</b>	<b>unit_tests package</b>	<b>19</b>
6.1	test_analyze module . . . . .	19
6.2	test_extractmicrosoftdocxtext module . . . . .	19
6.3	test_extractpdftext module . . . . .	19
6.4	test_outputkeywordtotext module . . . . .	19
6.5	test_pdfanddocxareadthesame module . . . . .	20
<b>7</b>	<b>behave_tests package</b>	<b>21</b>
7.1	tutorial module . . . . .	21
<b>8</b>	<b>Indices and tables</b>	<b>23</b>
	<b>Python Module Index</b>	<b>25</b>
	<b>Index</b>	<b>27</b>



**class** Keyword.**Keyword** (*nWord=""*, *nType=0*, *nSal=0*, *nFreq=0*, *nKeyscore=0*)

Bases: object

**summary:** Stores a specific keyword and it's associated information. The constructor accepts the word, type, salience, frequency and key score.

**classmethod issimilar** (*passedWord*)

summary: determines if the passed keyword is similar to (or exactly the same as) the main word in the class

**Parameters** **passedWord** (*str*) – word

**Returns** boolean value of True or False

**Return type** bool

**similarwordfrequency** ()

**Returns** the frequency of a similar word in a document

**Return type** int

**wordfrequency** ()

**Returns** the frequency value of a word

**Return type** int



---

### KeywordList module

---

**class** KeywordList.**KeywordList**

Bases: object

**Summary:** A list that contains keywords. The list also contains unique keyword value, keyword score, yules k score, yules score and a document score.

**calculateavgscores** ()

Summary: calculates a document's average score values.

**Returns** void

**existsinlist** (*keyword\_name*)

Summary: searches through the list of keywords and sees if any keywords shares the same Keyword.word.

**Parameters** **keyword\_name** (*str*) – The keyword

**Returns** returns true if a keyword with keyword\_name as Keyword.word exists in the list. False otherwise.

**Return type** bool

**getavgkeywordscore** ()

Summary: returns document's average keyword score.

**Returns** average keyword score

**Return type** int

**getdocumentscore** ()

Summary: Returns document's score.

**Returns** document score

**Return type** int

**getindexofword** (*keyword\_name*)

Summary: returns index of a Keyword in the list of Keywords

**Parameters** **key\_name** (*str*) – keyword

**Returns** keyword index

**Return type** int

**getkeywordscore** ()

Summary: returns document's keyword score.

**Returns** keyword score of document

**Return type** int

**getyulesiscore** ()

Summary: returns document's Yule's i score.

**Returns** Yule's I score

**Return type** int

**getyuleskscore** ()

Summary: returns document's Yule's k score.

**Returns** Yules K score

**Rytpe** int

**insertkeyword** (*keyword*)

Summary: inserts new Keyword into Keyword list

**Parameters** **keyword** ([Keyword](#)) – an instance of the class keyword

**Returns** void



### 3.1 common\_functions module

`common_functions.changefileextension` (*regfilename*)

Changes the file name string from .pdf to .txt.

**Parameters** `regfilename` (*str*) – name of regulatory file

**Returns** string with .pdf file extension

**Return type** `str`

`common_functions.cleantext` (*text\_list*)

Removes special characters from text

**Parameters** `text_list` (*List[str]*) – a text string

**Returns** `text_list` with no special chars

**Return type** `List[str]`

`common_functions.createkeywordfromgoogleapientity` (*entity*, *file\_text*)

Creates a Keyword from a single entity that is returned by the google API

**Parameters**

- **entity** (*Entity*) – Google API response entity object
- **file\_text** (*List[str]*) – entire text of file

**Returns** Populated Keyword object

**Return type** *Keyword*

`common_functions.extractkeywordfromtxt` (*filename*)

This function will extract keyword information from .txt file and place into KeywordList object

**Parameters** `file` (*str*) – location of .txt file

**Returns** keyword list in file

**Return type** *KeywordList*

`common_functions.extractmicrosoftdocxtext` (*file*, *testdownload\_folder=None*)

Extracts text from any “.docx” document and returns it.

**Parameters**

- **file** (*fileStorage*) – the file to save
- **testdownload\_folder** (*str*) – Specific download folder is necessary

**Returns** file’s text

**Return type** List[str]

`common_functions.extractpdftext` (*file*, *testdownload\_folder=None*, *RegDoc=False*)

Extracts Text from PDF document referenced in given file argument

**Parameters**

- **file** (*fileStorage*) – the PDF file to extract text from
- **testdownload\_folder** (*str*) – specific download folder if necessary
- **RegDoc** (*bool*) – flag specifying whether this is a user doc or a regulatory doc

**Returns** file’s text

**Return type** List[str]

`common_functions.generatebubblecsv` (*kw\_list*, *reg\_kw\_list*)

Creates a new csv file with all the keywords

**Parameters**

- **kw\_list** (*KeywordList*) – list of doc keywords
- **reg\_kw\_list** (*KeywordList*) – list of reg doc keywords

**Returns** void

`common_functions.geterrorpage` (*errtext='Unknown Error'*)

Populates error message with proper response and returns html

**Parameters** **errtext** (*str*) – text of error

**Returns** html page with error displayed

**Return type** str

`common_functions.getregulatorydoctext` (*filename*)

Looks in the RegulatoryDocuments folder for the file with the given file name and return’s its text as a list of string

**Parameters** **filename** (*str*) – name of regulatory file without file ending on it

**Returns** list of strings of length 1024 containing text of file

**Return type** List[str]

`common_functions.getscorepage` (*kw\_list*, *reg\_kw\_list*, *userdocwordcount*, *filename*, *regfilename*)

Returns html page that is populated with proper calculated Keyword, Comparison, and Yule’s scores.

**Parameters**

- **kw\_list** (*KeywordList*) – list of user document’s Keyword objects
- **reg\_kw\_list** (*KeywordList*) – list of regulatory document’s Keywords

**Returns** html page with scores displayed

**Return type** str

`common_functions.getwordfrequency(word, file_text)`

Determines frequency of the given word in the file's text

**Parameters**

- **word** (*str*) – Word to find frequency of
- **filetext** (*List[str]*) – list of string containing entire text of file

**Returns** frequency of word parameter in text

**Return type** int

`common_functions.homeCount()`

Initializes variables for logging session

**Returns** void

`common_functions.interpretexistingfile(regfilename)`

Parses, identifies keywords and analyzes content of chosen regulatory file document is being compares against.

**Parameters** **regfilename** (*str*) – name of regulatory file

**Returns** list of analyzed Keyword objects

**Return type** *KeywordList*

`common_functions.interpretfile(file, localuploadfolder)`

Parses uploaded file's text, identifies keywords, analyzes keywords, and returns a list of Keyword Objects

**Parameters**

- **file** (*fileStorage*) – file to be interpreted
- **localuploadfolder** (*str*) – Place to temporary store file so it can be read from

**Returns** list of file's Keywords

**Return type** *KeywordList*

`common_functions.kwhighestfrequencies(keyword_list, numtopkws=10)`

Returns the top 10 most frequent Keywords in the user's uploaded file

**Parameters** **keyword\_list** (*KeywordList*) – List of Keyword objects

**Returns** Keywords with highest frequencies

**Return type** *List[Keyword]*

`common_functions.kwhighestkeyscores(keyword_list)`

Returns ten Keywords with the highest Keyword scores

**Parameters** **keyword\_list** (*KeywordList*) – list of Keyword objects

**Returns** list of top keyword scores

**Return type** *List[Keyword]*

`common_functions.longstringtolist(longstring, strsize)`

This functions splits a long string "longstring" into strings of size "strsize" and returns a list of those strings.

**Parameters**

- **longstring** (*string*) – text of file

- **strsize** (*int*) – requested length of each string in created list of strings

**Returns** file text

**Return type** List[str]

`common_functions.outputkeywordtotext (keylist, download_folder='Documents/Keywords.txt')`

This function will write Keywords from an analyzed document to a .txt file

**Parameters** **keylist** (*KeywordList*) – list of document keywords

**Returns** void

`common_functions.plotkeywordfrequency (keyword_list1, keyword_list2, doc1name='doc1', doc2name='doc2')`

Plots keyword score of most frequently used keywords. Saves graph to “/Downloads” folder

**Parameters**

- **keyword\_list1** (*KeywordList*) – user document keywords
- **keyword\_list2** (*KeywordList*) – regulatory document keywords
- **doc1name** (*str*) – name of user document
- **doc2name** (*str*) – name of regulatory document

**Returns** void

`common_functions.plotkeywordsalience (keyword_list1, keyword_list2, doc1name='doc1', doc2name='doc2')`

Plots salience of most frequently used keywords. Pulls KWs from list1, compares against list2

**Parameters**

- **keyword\_list1** (*KeywordList*) – user KeywordList
- **keyword\_list2** (*KeywordList*) – regulatory KeywordList
- **doc1name** (*str*) – user document name
- **doc2name** (*str*) – regulatory document name

**Returns** void

`common_functions.plotkeywordscores (keyword_list1, keyword_list2, doc1name='doc1', doc2name='doc2')`

Plots keyword score of most frequently used keywords. Pulls KWs from list1, compares against list2

**Parameters**

- **keyword\_list1** (*KeywordList*) – user KeywordList
- **keyword\_list2** (*KeywordList*) – regulatory KeywordList
- **doc1name** (*str*) – user document name
- **doc2name** (*str*) – regulatory document name

**Returns** void

`common_functions.printStringList (textList)`

Helper function that prints a list of strings

**Parameters** **textList** (*List[str]*) – a text string

**Returns** void

`common_functions.printanalytics (filename, regfilename, keywordlist, regkeywordlist, calctime)`

prints the data passed in the argument to the ever-increasing file that contains data analytics information

**Parameters** `printstr` (*str*) – string to output to file

**Returns** void

`common_functions.savefile` (*file*, *download\_folder=None*)

Save's given file to /Downloads folder"

**Parameters**

- **file** (*fileStorage*) – the file to save
- **download\_folder** (*str*) – specific download folder if necessary

**Returns** void

`common_functions.splitintosize` (*file\_text*)

This function splits a list of keywords of any length into a list of keywords each of length specified by NUM\_SEND\_CHARS in 'applicationconfig.json'

**Parameters** `file_text` (*list*) – list of document's words

**Return** list `file_text`

`common_functions.stringlisttolonglongstring` (*string\_list*)

Helper function to turn list of string into one long long string

**Parameters** `string_list` (*List[str]*) – a string of text

**Returns** file's text

**Return type** long string

`common_functions.writeToConfig` (*key*, *value*)

Writes value into applicationconfig.json file

**Parameters**

- **key** – key
- **value** – value

**Returns** none

## 3.2 analyze\_functions module

`analyze_functions.calculatecomparisonscore` (*kw\_list*, *reg\_kw\_list*)

Summary: Compares the calculated scores of the two documents and generates value based on that comparison.

1. The top 10 Keywords with the highest frequency is gathered from the user document.
2. The top 10% of the regulatory document Keywords are gathered.
3. For the top 10 Keywords in the user document, if they are in the top 10% of words in the regulatory document, a value of '1' is added to a variable called *tempscore*.
4. *tempscore* / top 10% of reg doc keywords = the new *tempscore*
5. The final score that is returned: 100 - [abs(average keyword score of user doc - average keyword score of reg doc)] \* *tempscore*

**Parameters**

- **kw\_list** (*KeywordList*) – list of Keywords
- **reg\_kw\_list** (*KeywordList*) – list of Keywords

**Returns** comparison score of two documents

**Return type** float

`analyze_functions.calculatekeywordscore(kw_list, kw)`

Summary: calculate a keyword score for a single keyword. The current algorithm utilized is:  $[(\text{keyword salience} * \text{keyword frequency}) / (\text{total keywords})] * 1000$ . Since the salience and frequency of a particular keyword is important to the overall feel of a document, these values are used to calculate the score.

**Parameters**

- **kw\_list** (`KeywordList`) – all Keywords of a document.
- **kw** (`Keyword`) – keyword

**Returns** keyword score

**Return type** float

`analyze_functions.calculatescores(kw_list, file_text)`

Summary: function that calls `calculatekeywordscore()` and `calculateyulescore()` and inputs those values into `Keyword` and `KeywordList` respectively for a particular document.

**Parameters**

- **kw\_list** (`KeywordList`) – list of Keywords
- **file\_text** (`List[str]`) – Text of file

**Returns** void

`analyze_functions.calculateyulescore(file_text)`

Summary: calculates Yule's K/I scores for a given document. These scores are used to determine the lexical richness of a given document.

This function starts by ensuring that `file_text` is converted into a long string vice a list of strings to ensure accurate calculation of the scores. Then, the string is split into tokens via `tokenize()`. The Yule's K/I algorithm is implemented based on the tokens provided. If there is a 'Division by Zero' error, an exception will be raised and the default score value will be '-1'

**Parameters** **file\_text** (`List[str]`) – plain text of document

**Returns** Yules score of text file [Yule's K, Yule's I]

**Return type** float

**Raises** `ZeroDivisionError`

`analyze_functions.declarelogger()`

Summary: Declares logger for the current session. Logging statements are re-directed to a local logging file. The logging level is set to `DEBUG`.

`LOG_FILE_PATH = 'logging/Linguistic_Analyzer.log'`

`analyze_functions.identifykeywords(file_text)`

Summary: Calls the Google NLP API to extract Keyword information from text. The 'analyze entities' from the API is utilized. The information retained from the API is 'entity' (keyword) and the 'salience' value of a particular keyword.

Information regarding the Google NLP API can be found at: <https://cloud.google.com/natural-language/>

For use on a local machine: add `export API_KEY="your API key"` in `bash.profile` or whichever file contains environmental variable setup.

For use in AWS: enter 'API\_KEY' with key value in AWS configuration settings

*file\_text* contains the text of a particular document in a list of strings. The original idea here was concern that a long string of text would crash the app due to memory constraints. However, if document text is broken up and sent to the API as such, the analysis would not encompass the document in its entirety. Instead, the scores provided would be focused on each ‘chunk’ of text. Therefore, analysis of an entire document would be inaccurate. The list of strings idea here has remained, but the ‘chunk’ size for *file\_text* can be configured in `/applicationconfig.json`. Default settings allow for a single string text input of a document into the API.

For each entity identified by the API, `commonfunctions.createkeywordfromgoogleapientity()` is used to extract the information from the *entities* dictionary variable and places it into a *Keyword*. The returned Keyword is then placed into the *KeywordList* object via `KeywordList.insertkeyword()`.

**Parameters** `file_text` (*List[str]*) – text of document

**Returns** *KeywordList* object

**Return type** *KeywordList*

**Raises** *Exception*

`analyze_functions.tokenize(tokenStr)`

Summary: Splits up string into individual tokens.

**Parameters** `tokenStr` (*str*) – a string of words

**Returns** *tokens*

**Return type** *list*





## CHAPTER 4

---

### analyze module

---

`analyze.analyzeText` (*fileText*)

**Parameters** `fileText` (*str*) – text of fileText

**Returns** file text

**Return type** `str`

`analyze.checkSimilarity` (*fileText*)

**Parameters** `fileText` (*str*) – text of file

**Returns** pass or fail

**Return type** `bool`

`analyze.createObject` (*fileText*)

**Parameters** `fileText` (*str*) – text of file

**Returns** pass or fail

**Return type** `bool`

`analyze.scrapeText` (*fileText*)

**Parameters** `fileText` (*str*) – text of file

**Returns** pass or fail

**Return type** `bool`



`application.analyze()`

Receives uploaded document and comparison document choice and executes logic to compare them.

**Returns** Information regarding the uploaded document's similarity to regulatory document

**Return type** html

`application.bubbletest()`

Page for testing

**Returns** Test page

**Return type** html

`application.comparisoninfo()`

Comparison Information

**Returns** graph html page that describes the Linguistic Analyzer's Comparison Score

**Return type** html

`application.getapplicationconfig()`

Returns json application config file

**Returns** applicationconfig.json

**Return type** json file

`application.getbackgroundimg()`

Returns png image of file at

**Returns** graph

**Return type** png

`application.getbackgroundwordsimg()`

Returns png image of a graph of words background

**Returns** graph

**Return type** png

`application.getcsvkeywords()`

Returns csvkeywords.csv

**Returns** csvkeywords keyword file

**Return type** csv

`application.getdocumentationhome()`

Returns index page nested in Documentation/\_build/html which is the home page for our Sphinx-generated documentation

**Returns** html text

`application.getkwfreeqimage()`

Returns Keyword frequency graph

**Returns** graph

**Return type** png

`application.getkwsalienceimage()`

Returns png image of a graph of top salience keywords

**Returns** graph

**Return type** png

`application.getkwcoresimage()`

Returns png image of a graph of keyword scores

**Returns** graph

**Return type** png

`application.getlinguisticalyzerlog()`

Returns LinguisticAnalyzer.log

**Returns** log file

**Return type** log

`application.getregdockws()`

Returns Reg\_Keywords.txt

**Returns** regulatory doc keyword file

**Return type** txt

`application.gettestkeywords()`

Returns test\_keywords.csv

**Returns** test\_keywords doc keyword file

**Return type** csv

`application.getuserdockws()`

Returns Keywords.txt

**Returns** keyword file

**Return type** txt

`application.indexjs()`

Page for testing

**Returns** Test page

**Return type** html

`application.keywordbubblechart()`

Returns bubble chart html page

**Returns** bubble chart html page

**Return type** html

`application.main()`

Home page of the Linguistic Analyzer API

**Returns** Home page

**Return type** html

`application.newregdoc()`

Adds new regulatory document

**Returns** none

**Return type** none

`application.project()`

Returns an html page containing details about the Linguistic Analyzer project.

**Returns** Home page

**Return type** html

`application.resource_path(relative_path)`

Summary: Function to determine correct file path of directories for use within an IDE or executable.

**Parameters** `relative_path` (*str*) – the path of a directory relative to a local environment

**Returns** base\_path in relation to executable environment and relative\_path of local environment

**Return type** string

`application.reusablebubble()`

Page for testing

**Returns** Test page

**Return type** html

`application.reusablebubblejs()`

Page for testing

**Returns** Test page

**Return type** html

`application.yulesinfo()`

Yule's Info

**Returns** Page that describes Yule's k and Yule's i algorithms

**Return type** html



### 6.1 test\_analyze module

```
class unit_tests.test_analyze.TestAnalyze (methodName='runTest')  
    Bases: unittest.case.TestCase  
  
    test_analyze ()  
        Summary: Tests the Analyze() function
```

### 6.2 test\_extractmicrosoftdocxtext module

```
class unit_tests.test_extractmicrosoftdocxtext.TestExtractmicrosoftdocxtext (methodName='runTest')  
    Bases: unittest.case.TestCase  
  
    test_extractmicrosoftdocxtext ()  
        Summary: Tests the extractmicrosoftdoctet() function
```

### 6.3 test\_extractpdftext module

```
class unit_tests.test_extractpdftext.TestExtractpdftext (methodName='runTest')  
    Bases: unittest.case.TestCase  
  
    test_extractpdftext ()  
        Summary: Tests the extractpdftext() function
```

### 6.4 test\_outputkeywordtotext module

```
class unit_tests.test_outputkeywordtotext.TestOutputkeywordtotext (methodName='runTest')  
    Bases: unittest.case.TestCase
```

```
test_outputkeywordtotext()
```

## 6.5 test\_pdfanddocxarereadthesame module

```
class unit_tests.test_pdfanddocxarereadthesame.TestEnsurepdfanddocxarereadthesame (methodName)
    Bases: unittest.case.TestCase

    test_ensurepdfanddocarereadthesame()
        Summary: tests whether extractpdftext() and extractdocxtext() return the same exact information when
        given the same document in different formats
```



---

behave\_tests package

---

## 7.1 tutorial module

```
behave_tests.tutorial.steps.tutorial.step_impl (context)  
    @type context: behave.runner.Context
```



## CHAPTER 8

---

### Indices and tables

---

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



### a

`analyze`, [13](#)  
`analyze_functions`, [9](#)  
`application`, [15](#)

### b

`behave_tests.tutorial.steps.tutorial`,  
[21](#)

### c

`common_functions`, [5](#)

### k

`Keyword`, [1](#)  
`KeywordList`, [3](#)

### u

`unit_tests.test_analyze`, [19](#)  
`unit_tests.test_extractmicrosoftdocxtext`,  
[19](#)  
`unit_tests.test_extractpdftext`, [19](#)  
`unit_tests.test_outputkeywordtotext`, [19](#)  
`unit_tests.test_pdfanddocxarereadthesame`,  
[20](#)



**A**

analyze (module), 13  
analyze() (in module application), 15  
analyze\_functions (module), 9  
analyzeText() (in module analyze), 13  
application (module), 15

**B**

behave\_tests.tutorial.steps.tutorial (module), 21  
bubbletest() (in module application), 15

**C**

calculateavgscores() (KeywordList.KeywordList method), 3  
calculatecomparisonscore() (in module analyze\_functions), 9  
calculatekeywordscore() (in module analyze\_functions), 10  
calculatescores() (in module analyze\_functions), 10  
calculateyulessscore() (in module analyze\_functions), 10  
changefileextension() (in module common\_functions), 5  
checkSimilarity() (in module analyze), 13  
cleantext() (in module common\_functions), 5  
common\_functions (module), 5  
comparisoninfo() (in module application), 15  
createkeywordfromgoogleapientity() (in module common\_functions), 5  
createObjects() (in module analyze), 13

**D**

declarelogger() (in module analyze\_functions), 10

**E**

existsinlist() (KeywordList.KeywordList method), 3  
extractkeywordfromtxt() (in module common\_functions), 5  
extractmicrosoftdocxtext() (in module common\_functions), 6  
extractpdftext() (in module common\_functions), 6

**G**

generatebubblecsv() (in module common\_functions), 6  
getapplicationconfig() (in module application), 15  
getavgkeywordscore() (KeywordList.KeywordList method), 3  
getbackgroundimg() (in module application), 15  
getbackgroundwordsimg() (in module application), 15  
getcsvkeywords() (in module application), 16  
getdocumentationhome() (in module application), 16  
getdocumentscore() (KeywordList.KeywordList method), 3  
geterrorpage() (in module common\_functions), 6  
getindexofword() (KeywordList.KeywordList method), 3  
getkeywordscore() (KeywordList.KeywordList method), 4  
getkwfreeqimage() (in module application), 16  
getkwsalienceimage() (in module application), 16  
getkwscoresimage() (in module application), 16  
getlinguisticalyzerlog() (in module application), 16  
getregdockws() (in module application), 16  
getregulatorydoctext() (in module common\_functions), 6  
getscorepage() (in module common\_functions), 6  
gettestkeywords() (in module application), 16  
getuserdockws() (in module application), 16  
getwordfrequency() (in module common\_functions), 7  
getyulesiscore() (KeywordList.KeywordList method), 4  
getyuleskscore() (KeywordList.KeywordList method), 4

**H**

homeCount() (in module common\_functions), 7

**I**

identifykeywords() (in module analyze\_functions), 10  
indexjs() (in module application), 16  
insertkeyword() (KeywordList.KeywordList method), 4  
interpretexistingfile() (in module common\_functions), 7  
interpretfile() (in module common\_functions), 7  
issimilar() (Keyword.Keyword class method), 1

## K

Keyword (class in Keyword), 1  
 Keyword (module), 1  
 keywordbubblechart() (in module application), 17  
 KeywordList (class in KeywordList), 3  
 KeywordList (module), 3  
 kwhighestfrequencies() (in module common\_functions), 7  
 kwhighestkeyscores() (in module common\_functions), 7

## L

longstringtostringlist() (in module common\_functions), 7

## M

main() (in module application), 17

## N

newregdoc() (in module application), 17

## O

outputkeywordtotext() (in module common\_functions), 8

## P

plotkeywordfrequency() (in module common\_functions), 8  
 plotkeywordsalience() (in module common\_functions), 8  
 plotkeywordscores() (in module common\_functions), 8  
 printanalytics() (in module common\_functions), 8  
 printStringList() (in module common\_functions), 8  
 project() (in module application), 17

## R

resource\_path() (in module application), 17  
 reusablebubble() (in module application), 17  
 reusablebubblejs() (in module application), 17

## S

savefile() (in module common\_functions), 9  
 scrapeText() (in module analyze), 13  
 similarwordfrequency() (Keyword.Keyword method), 1  
 splitintotext() (in module common\_functions), 9  
 step\_impl() (in module have\_tests.tutorial.steps.tutorial), 21  
 stringlisttolonglongstring() (in module common\_functions), 9

## T

test\_analyze() (unit\_tests.test\_analyze.TestAnalyze method), 19  
 test\_ensurepdfanddocarereadthesame() (unit\_tests.test\_pdfanddocarereadthesame.TestEnsurepdfanddocarereadthesame method), 20

test\_extractmicrosoftdocxtext() (unit\_tests.test\_extractmicrosoftdocxtext.TestExtractmicrosoftdocxtext method), 19  
 test\_extractpdftext() (unit\_tests.test\_extractpdftext.TestExtractpdftext method), 19  
 test\_outputkeywordtotext() (unit\_tests.test\_outputkeywordtotext.TestOutputkeywordtotext method), 19  
 TestAnalyze (class in unit\_tests.test\_analyze), 19  
 TestEnsurepdfanddocarereadthesame (class in unit\_tests.test\_pdfanddocarereadthesame), 20  
 TestExtractmicrosoftdocxtext (class in unit\_tests.test\_extractmicrosoftdocxtext), 19  
 TestExtractpdftext (class in unit\_tests.test\_extractpdftext), 19  
 TestOutputkeywordtotext (class in unit\_tests.test\_outputkeywordtotext), 19  
 tokenize() (in module analyze\_functions), 11

## U

unit\_tests.test\_analyze (module), 19  
 unit\_tests.test\_extractmicrosoftdocxtext (module), 19  
 unit\_tests.test\_extractpdftext (module), 19  
 unit\_tests.test\_outputkeywordtotext (module), 19  
 unit\_tests.test\_pdfanddocarereadthesame (module), 20

## W

wordfrequency() (Keyword.Keyword method), 1  
 writeToConfig() (in module common\_functions), 9

## Y

yulesinfo() (in module application), 17