pylightxl

Release 2019

Feb 01, 2023

Contents

1 High-Level Feature Summary 3

2 Limitations 5
- 2.1 Installation 5
- 2.2 Quick Start Guide 6
- 2.3 Source Code Documentation 11
- 2.4 Example Solutions 17
- 2.5 Revision Log 18
- 2.6 License 22
- 2.7 Contributor Code of Conduct 23

3 Support Content Creator 25

Index 27

Fig. 1: pylightxl pypi | github

A light weight, zero dependency (only standard libs used), to the point (no bells and whistles) Microsoft Excel reader/writer python 2.7.18 - 3+ library. Although there are several excellent read/write options out there (python-excel.org or excelpython.org) pylightxl focused on the following key features:

- **Zero non-standard library dependencies!**
  - No compatibility/version control issues.
- **Python2.7.18 to Python3+ support for life!**
  - Don't worry about which python version you are using, pylightxl will support it for life
- **Light-weight single source code file**
  - Want your project remain truly dependent-less? Copy the single source file into your project without any extra dependency issues or setup.
  - Do you struggle with other libraries weighing your projects down due to their very large size? Pylightxl's single source file size and zero dependency will not weight your project down (preferable for django apps)
  - Do you struggle with `pyinstaller` or other `exe` wrappers failing to build or building to very large packages? Pylightxl will not cause any build errors and will not add to your build size since it has zero dependencies and a small lib size.
  - Do you struggle with download restrictions at your company? Copy the entire pylightxl source from 1 single file and use it in your project.
- **100% test-driven development for highest reliability/maintainability that aims for 100% coverage on all supported versions**
  - Pylightxl aims to test all of its features, however unforeseen edge cases can occur when receiving excel files created by non-microsoft excel. We actively monitor issues to add support for these edge cases should they arise.
- **API aimed to be user friendly and intuitive and well documented. Structure: database > worksheet > indexing**
  - `db.ws('Sheet1').index(row=1,col=2)` or `db.ws('Sheet1').address(address='B1')`
  - `db.ws('Sheet1').row(1)` or `db.ws('Sheet1').col(1)`

CHAPTER 1

High-Level Feature Summary

- **Reader**
  - supports Microsoft Excel 2004+ files (`.xlsx`, `.xlsm`) and `.csv` files
  - read files via str path, pathlib path or file objects
  - read all or selective sheets
  - read type converted cell value (string, int, float), formula, comments, and named ranges
- **Database**
  - call cell value by row/col ID, excel address, or range
  - call an entire row/col or a semi-structured table based on user-defined headers
- **Writer**
  - write to new excel file (write excel files without having excel on your machine)
  - write to existing excel files (see limitations below)

CHAPTER 2

Limitations

Although every effort was made to support a variety of users, the following limitations should be read carefully:

- Does not support `.xls` files (Microsoft Excel 2003 and older files)
- Writer does not support anything other than cell data (no graphs, images, macros, formatting)
- Does not support worksheet cell data more than 536,870,912 cells (32-bit list limitation), please use 64-bit if more data storage is required.

2.1 Installation

There are several ways to use `pylightxl`. The easiest download is through `pip` however we understand that certain workplaces may not allow externally downloaded content, therefore we made it easy to copy/paste the source code needed to get going as well. Please see License terms of agreement: *License*

2.1.1 pip install via pypi

Download via Python Package Installer the latest official release:

```
pip install pylightxl
```

2.1.2 pip install via github

Download via github the latest master branch release:

```
pip install git+https://github.com/PydPiper/pylightxl.git
```

2.1.3 Git Clone

Download via github clone:

```
git clone https://github.com/PydPiper/pylightxl.git
```

2.1.4 Download Source Files

Download via github: https://github.com/PydPiper/pylightxl/archive/master.zip

2.1.5 Copy Source Files

Create a copy of the entire library that the user can copy directly into a project, a virtual environment, or into the python/lib/site-packages folder for general use.

1.) Create a folder `pylightxl`

2.) Create the following files within the `pylightxl` folder:

```
pylightxl
    1- __init__.py
    2- pylightxl.py
```

3.) Populate the files with their respective source code contents:

3.1) File1: __init__.py

3.2) File2: pylightxl.py

2.2 Quick Start Guide

Get up and running in less than 5 minutes with pylightxl!

2.2.1 Read/Write CSV File

Read a csv file with contents:

```
import pylightxl as xl

# set the delimiter of the CSV to be the value of your choosing
# set the default worksheet to write the read in CSV data to
db = xl.readcsv(fn='input.csv', delimiter='/', ws='sh2')

# make modifications to it then,
# now write it back out as a csv; or as an excel file, see xl.writexl()
xl.writecsv(db=db, fn='new.csv', ws=('sh2'), delimiter=',')
```

2.2.2 Read Excel File

```
import pylightxl as xl

# readxl returns a pylightxl database that holds all worksheets and its data
db = xl.readxl(fn='folder1/folder2/excelfile.xlsx')

# pylightxl also supports pathlib as well
my_pathlib = pathlib.Path('folder1/folder2/excelfile.xlsx')
db = xl.readxl(my_pathlib)

# pylightxl also supports file-like objects for django users
with open('excelfile.xlsx', 'rb') as f:
    db = xl.readxl(f)

# read only selective sheetnames
db = xl.readxl(fn='folder1/folder2/excelfile.xlsx', ws=('Sheet1','Sheet3'))

# return all sheetnames
db.ws_names
>>> ['Sheet1', 'Sheet3']
```

2.2.3 Access Worksheet and Cell Data

The following example assumes `excelfile.xlsx` contains a worksheet named `Sheet1` and it has the following cell content:

	A	B	C
1	10	20	
2		30	40

Via Cell Address

```
db.ws(ws='Sheet1').address(address='A1')
>>> 10
# access the cell's formula (if there is one)
db.ws(ws='Sheet1').address(address='A1', output='f')
>>> ''
# access the cell's comment (if there is one)
db.ws(ws='Sheet1').address(address='A1', output='c')
>>> 'this is a comment on cell A1!'
# note index a empty cell will return an empty string
db.ws(ws='Sheet1').address(address='A100')
>>> ''
# however default empty value can be overwritten for each worksheet
db.ws(ws='Sheet1').set_emptycell(val=0)
db.ws(ws='Sheet1').address(address='A100')
>>> 0
```

Via Cell Index

```
db.ws(ws='Sheet1').index(row=1, col=2)
>>> 20
# access the cell's formula (if there is one)
db.ws(ws='Sheet1').index(row=1, col=2, output='f')
>>> '=A1+10'
# note index a empty cell will return an empty string
db.ws(ws='Sheet1').index(row=100, col=1)
>>> ''
# however default empty value can be overwritten for each worksheet
db.ws(ws='Sheet1').set_emptycell(val=0)
db.ws(ws='Sheet1').index(row=100, col=1)
>>> 0
```

Via Cell Range

```
db.ws(ws='Sheet1').range(address='A1')
>>> 10
db.ws(ws='Sheet1').range(address='A1:C2')
>>> [[10, 20, ''], ['', 30, 40]]
# get the range's formulas
db.ws(ws='Sheet1').range(address='A1:B1', output='f')
>>> [['=10', '=A1+10']]
# update a range with a single value
db.ws(ws='Sheet1').update_range(address='A1:B1', val=10)
```

Get entire row or column

```
db.ws(ws='Sheet1').row(row=1)
>>> [10,20,'']

db.ws(ws='Sheet1').col(col=1)
>>> [10,'']
```

Iterate through rows/cols

```
for row in db.ws(ws='Sheet1').rows:
    print(row)

>>> [10,20,'']
>>> ['',30,40]

for col in db.ws(ws='Sheet1').cols:
    print(col)

>>> [10,'']
>>> [20,30]
>>> ['',40]
```

Update Cell Value

```
db.ws(ws='Sheet1').address(address='A1')
>>> 10
db.ws(ws='Sheet1').update_address(address='A1', val=100)
db.ws(ws='Sheet1').address(address='A1')
>>> 100

db.ws(ws='Sheet1').update_index(row=1, col=1, val=10)
db.ws(ws='Sheet1').index(row=1, col=1)
>>> 10
```

Update Cell Formula

Same as update cell value except the entry must begin with a equal sign "="

Note: updating a cell formula will clear the previously read in cell value. Formulas will not calculate their cell value until the excel file is opened.

```
db.ws(ws='Sheet1').update_address(address='A1', val='=B1+100')
db.ws(ws='Sheet1').update_index(row=1, col=1, val='=B1+100')
```

Get Named Ranges

```
# define a named range
db.add_nr(name='Table1', ws='Sheet1', address='A1:B2')
# get the contents of a named ranges
db.nr(name='Table1')
>>> [[10, 20], ['', 30]]
# find the location of a named range
db.nr_loc(name='Table1')
>>> ['Sheet1','A1:B2']
# update the value of a named range
db.update_nr(name='Table1', val=10)
# see all existing named ranges
db.nr_names
>>> {'Table1': 'Sheet1!A1:B2'}
# remove a named range
db.remove_nr(name='Table1')
```

Get row/col based on key-value

Note: key is type sensitive

```
# lets say we would like to return the column that has a cell value = 20 in row=1
db.ws(ws='Sheet1').keycol(key=20, keyindex=1)
>>> [20,30]

# we can also specify a custom keyindex (not just row=1), note that we now are␣
↪matched based on row=2
db.ws(ws='Sheet1').keycol(key=30, keyindex=2)
```

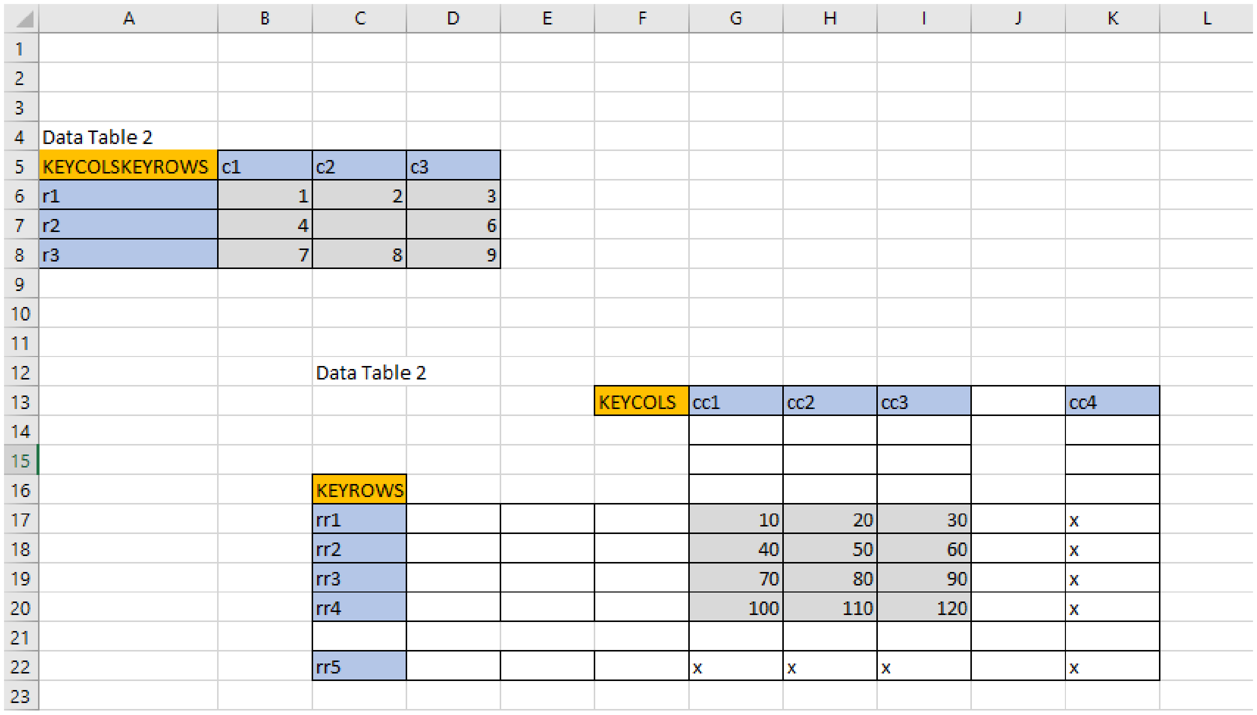
(continues on next page)

(continued from previous page)

```
>>> [20,30]

# similarly done for keyrow with keyindex=1 (look fora match in col=1)
db.ws(ws='Sheet1').keyrow(key='', keyindex=1)
>>> ['',30,40]
```

2.2.4 Read Semi-Structured Data



	A	B	C	D	E	F	G	H	I	J	K	L
1												
2												
3												
4	Data Table 2											
5	KEYCOLSKEYROWS	c1	c2	c3								
6	r1	1	2	3								
7	r2	4		6								
8	r3	7	8	9								
9												
10												
11												
12			Data Table 2									
13						KEYCOLS	cc1	cc2	cc3		cc4	
14												
15												
16			KEYROWS									
17			rr1				10	20	30		x	
18			rr2				40	50	60		x	
19			rr3				70	80	90		x	
20			rr4				100	110	120		x	
21												
22			rr5				x	x	x		x	
23												

- note that ssd function takes any key-word argument as your KEYROWS/KEYCOLS flag
- multiple tables are read the same way as you would read a book. Top left-to-right, then down

```
import pylightxl
db = pylightxl.readxl(fn='Book1.xlsx')

# request a semi-structured data (ssd) output
ssd = db.ws(ws='Sheet1').ssd(keycols="KEYCOLS", keyrows="KEYROWS")

ssd[0]
>>> {'keyrows': ['r1', 'r2', 'r3'], 'keycols': ['c1', 'c2', 'c3'], 'data': [[1, 2, 3],
↪ [4, '', 6], [7, 8, 9]]}
ssd[1]
>>> {'keyrows': ['rr1', 'rr2', 'rr3', 'rr4'], 'keycols': ['cc1', 'cc2', 'cc3'], 'data
↪': [[10, 20, 30], [40, 50, 60], [70, 80, 90], [100, 110, 120]]}
```

2.2.5 Write out a pylightxl.Database as an excel file

Pylightxl support excel writing without having excel installed on the machine. However it is not without its limitations. The writer only supports cell data writing (ie.: does not support graphs, formatting, images, macros, etc) simply just strings/numbers/equations in cells.

Note that equations typed by the user will not calculate for its value until the excel sheet is opened in excel.

```
import pylightxl as xl

# read in an existing worksheet and change values of its cells (same worksheet as␣
↪above)
db = xl.readxl(fn='excelfile.xlsx')
# overwrite existing number value
db.ws(ws='Sheet1').index(row=1, col=1)
>>> 10
db.ws(ws='Sheet1').update_index(row=1, col=1, val=100)
db.ws(ws='Sheet1').index(row=1, col=1)
>>> 100
# write text
db.ws(ws='Sheet1').update_index(row=1, col=2, val='twenty')
# write equations
db.ws(ws='Sheet1').update_address(address='A3', val='=A1')

xl.writexl(db=db, fn='updated.xlsx')
```

2.2.6 Write a new excel file from python data

For new python data that did not come from an existing excel speadsheet.

```
import pylightxl as xl

# take this list for example as our input data that we want to put in column A
mydata = [10,20,30,40]

# create a blank db
db = xl.Database()

# add a blank worksheet to the db
db.add_ws(ws="Sheet1")

# loop to add our data to the worksheet
for row_id, data in enumerate(mydata, start=1)
    db.ws(ws="Sheet1").update_index(row=row_id, col=1, val=data)

# write out the db
xl.writexl(db=db, fn="output.xlsx")
```

2.3 Source Code Documentation

2.3.1 readxl

`pylightxl.pylightxl.`**`readxl`**`(`*fn*, *ws=None*`)`

Reads an xlsx or xlsm file and returns a pylightxl database

Parameters

- **fn** (*Union[str, pathlib.Path]*) – Excel file path, also supports Pathlib.Path object, as well as file-like object from with/open

- **ws** (*Union[str,List[str]], optional*) – sheetnames to read into the database, if not specified - all sheets are read entry support single ws name (ex: ws='sh1') or multi (ex: ws=['sh1', 'sh2']), defaults to None

Returns pylightxl Database

Return type *Database*

2.3.2 writexl

`pylightxl.pylightxl.`**`writexl`**(*db*, *fn*)

Writes an excel file from pylightxl.Database

Parameters

- **db** (`Database`) – database contains sheetnames, and their data
- **fn** (*Union[str, pathlib.path]*) – file output path

2.3.3 database

Database Class

class `pylightxl.pylightxl.`**`Database`**

`add_nr`(*name*, *ws*, *address*)

Add a NamedRange to the database. There can not be duplicate name or addresses. A named range that overlaps either the name or address will overwrite the database's existing NamedRange

Parameters

- **name** (*str*) – NamedRange name
- **ws** (*str*) – worksheet name
- **address** (*str*) – range of address (single cell ex: "A1", range ex: "A1:B4")

`add_ws`(*ws*, *data=None*)

Logs worksheet name and its data in the database

Parameters

- **ws** (*str*) – worksheet name
- **data** (*dict, optional*) – dictionary of worksheet cell values (ex: {'A1': {'v':10,'f':'','s':'', 'c': ''}, 'A2': {'v':20,'f':'','s':'', 'c': ''}}), defaults to None

`nr`(*name*, *formula=False*, *output='v'*)

Returns the contents of a name range in a nest list form [row][col]

Parameters

- **name** (*str*) – NamedRange name
- **formula** (*bool, optional*) – flag to return the formula of this cell, defaults to False
- **output** (*str, optional*) – output request "v" for value, "f" for formula, "c" for comment, defaults to 'v'

Returns nest list form [row][col]

Return type List[list]

`nr_loc`(*name*)
Returns the worksheet and address loction of a named range

Parameters **`name`** (*`str`*) – NamedRange name

Returns [worksheet, address]

Return type List[str]

`nr_names`
Returns the dictionary of named ranges ex: {unique_name: unique_address, . . . }

Returns {unique_name: unique_address, . . . }

Return type Dict[str, str]

`remove_nr`(*name*)
Removes a Named Range from the database

Parameters **`name`** (*`str`*) – NamedRange name

`remove_ws`(*ws*)
Removes a worksheet and its data from the database

Parameters **`ws`** (*`str`*) – worksheet name

`rename_ws`(*old*, *new*)
Renames an existing worksheet. Caution, renaming to an existing new worksheet name will overwrite

Parameters

- **`old`** (*`str`*) – old name
- **`new`** (*`str`*) – new name

`set_emptycell`(*val*)
Custom definition for how pylightxl returns an empty cell

Parameters **`val`** (*`Union[str, int, float]`*) – (default=") empty cell value

`update_nr`(*name*, *val*)
Updates a NamedRange with a single value. Raises UserWarning if name not in workbook.

Parameters

- **`name`** (*`str`*) – NamedRange name
- **`val`** (*`Union[int, float, str]`*) – cell value; equations are string and must being with "="

`ws`(*ws*)
Indexes worksheets within the database

Parameters **`ws`** (*`str`*) – worksheet name

Returns pylightxl.Database.Worksheet class object

Return type *Worksheet*

`ws_names`
Returns a list of database stored worksheet names

Returns list of worksheet names

Return type List[str]

Worksheet Class

class `pylightxl.pylightxl.`**`Worksheet`**(*data=None*)

`address`(*address*, *formula=False*, *output='v'*)

Takes an excel address and returns the worksheet stored value

Parameters

- **`address`** (*`str`*) – Excel address (ex: "A1")
- **`formula`** (*`bool, optional`*) – flag to return the formula of this cell, defaults to False
- **`output`** (*`str, optional`*) – output request "v" for value, "f" for formula, "c" for comment, defaults to 'v'

Returns cell value

Return type Union[int, float, str, bool]

`col`(*col*, *formula=False*, *output='v'*)

Takes a col index input and returns a list of cell data

Parameters

- **`col`** (*`int`*) – col index (start at 1 that corresponds to column "A")
- **`formula`** (*`bool, optional`*) – flag to return the formula of this cell, defaults to False
- **`output`** (*`str, optional`*) – output request "v" for value, "f" for formula, "c" for comment, defaults to 'v'

Returns list of cell data

Return type List[Union[int, float, str, bool]]

`cols`

Returns a list of cols that can be iterated through

Returns list of cols-lists (ex: [[11,21],[12,22],[13,23]] for 2 rows with 3 columns of data

Return type Iterable[List[Union[int, float, str, bool]]]

`index`(*row*, *col*, *formula=False*, *output='v'*)

Takes an excel row and col starting at index 1 and returns the worksheet stored value

Parameters

- **`row`** (*`int`*) – row index (starting at 1)
- **`col`** (*`int`*) – col index (start at 1 that corresponds to column "A")
- **`formula`** (*`bool, optional`*) – flag to return the formula of this cell, defaults to False
- **`output`** (*`str, optional`*) – output request "v" for value, "f" for formula, "c" for comment, defaults to 'v'

Returns cell value

Return type Union[int, float, str, bool]

`keycol`(*key*, *keyindex=1*)

Takes a column key value (value of any cell within keyindex row) and returns the entire column, no match returns an empty list

Parameters

- **key** (*Union[str,int,float,bool]*) – any cell value within keyindex row (type sensitive)
- **keyindex** (*int, optional*) – option keyrow override. Must be >0 and smaller than worksheet size, defaults to 1

Returns list of the entire matched key column data (only first match is returned)

Return type List[Union[str,int,float,bool]]

keyrow (*key*, *keyindex=1*)

Takes a row key value (value of any cell within keyindex col) and returns the entire row, no match returns an empty list

Parameters

- **key** (*Union[str,int,float,bool]*) – any cell value within keyindex col (type sensitive)
- **keyindex** (*int, optional*) – option keyrow override. Must be >0 and smaller than worksheet size, defaults to 1

Returns list of the entire matched key row data (only first match is returned)

Return type List[Union[str,int,float,bool]]

range (*address*, *formula=False*, *output='v'*)

Takes a range (ex: "A1:A2") and returns a nested list [row][col]

Parameters

- **address** (*str*) – cell range (ex: "A1:A2", or "A1")
- **formula** (*bool, optional*) – returns the values if false, or formulas if true of cells, defaults to False
- **output** (*str, optional*) – output request "v" for value, "f" for formula, "c" for comment, defaults to 'v'

Returns nested list [row][col] regardless if range is a single cell or a range

Return type _type_

row (*row*, *formula=False*, *output='v'*)

Takes a row index input and returns a list of cell data

Parameters

- **row** (*int*) – row index (starting at 1)
- **formula** (*bool, optional*) – flag to return the formula of this cell, defaults to False
- **output** (*str, optional*) – output request "v" for value, "f" for formula, "c" for comment, defaults to 'v'

Returns list of cell data

Return type List[Union[int, float, str, bool]]

rows

Returns a list of rows that can be iterated through

Returns list of rows-lists (ex: [[11,12,13],[21,22,23]] for 2 rows with 3 columns of data

Return type Iterable[List[Union[int, float, str, bool]]]

set_emptycell (*val*)
Custom definition for how pylightxl returns an empty cell

Parameters **val** (*Union[int, float, str]*) – (default=") empty cell value

size
Returns the size of the worksheet (row/col)

Returns list of [maxrow, maxcol]

Return type List[int]

ssd (*keyrows='KEYROWS', keycols='KEYCOLS'*)
Runs through the worksheet and looks for "KEYROWS" and "KEYCOLS" flags in each cell to identify the start of a semi-structured data. A data table is read until an empty header is found by row or column. The search supports multiple tables.

Parameters

- **keyrows** (*str, optional*) – a flag to indicate the start of keyrow's cells below are read until an empty cell is reached, defaults to 'KEYROWS'
- **keycols** (*str, optional*) – a flag to indicate the start of keycol's cells to the right are read until an empty cell is reached, defaults to 'KEYCOLS'

Returns list of data dict in the form of [{'keyrows': [], 'keycols': [], 'data': [[], . . .]}, {. . . },]

Return type List[Dict[str,list]]

update_address (*address, val*)
Update worksheet data via address

Parameters

- **address** (*str*) – excel address (ex: "A1")
- **val** (*Union[int, float, str, bool]*) – cell value; equations are strings and must begin with "="

update_index (*row, col, val*)
Update worksheet data via index

Parameters

- **row** (*int*) – row index
- **col** (*int*) – column index
- **val** (*Union[int, float, str, bool]*) – cell value; equations are strings and must begin with "="

update_range (*address, val*)
Update worksheet data via address range with a single value

Parameters

- **address** (*str*) – excel address (ex: "A1:B3")
- **val** (*Union[int, float, str, bool]*) – cell value; equations are strings and must begin with "="

Support Functions

pylightxl.pylightxl.**utility_address2index** (*address*)
Convert excel address to row/col index

Parameters **`address`** (*`str`*) – Excel address (ex: “A1”)

Returns list of [row, col]

Return type List[int]

`pylightxl.pylightxl.`**`utility_index2address`**(*row*, *col*)

Converts index row/col to excel address

Parameters

- **`row`** (*`int`*) – row index (starting at 1)
- **`col`** (*`int`*) – col index (start at 1 that corresponds to column “A”)

Returns str excel address

Return type str

`pylightxl.pylightxl.`**`utility_columnletter2num`**(*text*)

Takes excel column header string and returns the equivalent column count

Parameters **`text`** (*`str`*) – excel column (ex: ‘AAA’ will return 703)

Returns int of column count

Return type int

`pylightxl.pylightxl.`**`utility_num2columnletters`**(*num*)

Takes a column number and converts it to the equivalent excel column letters

Parameters **`num`** (*`int`*) – column number

Returns excel column letters

Return type str

2.4 Example Solutions

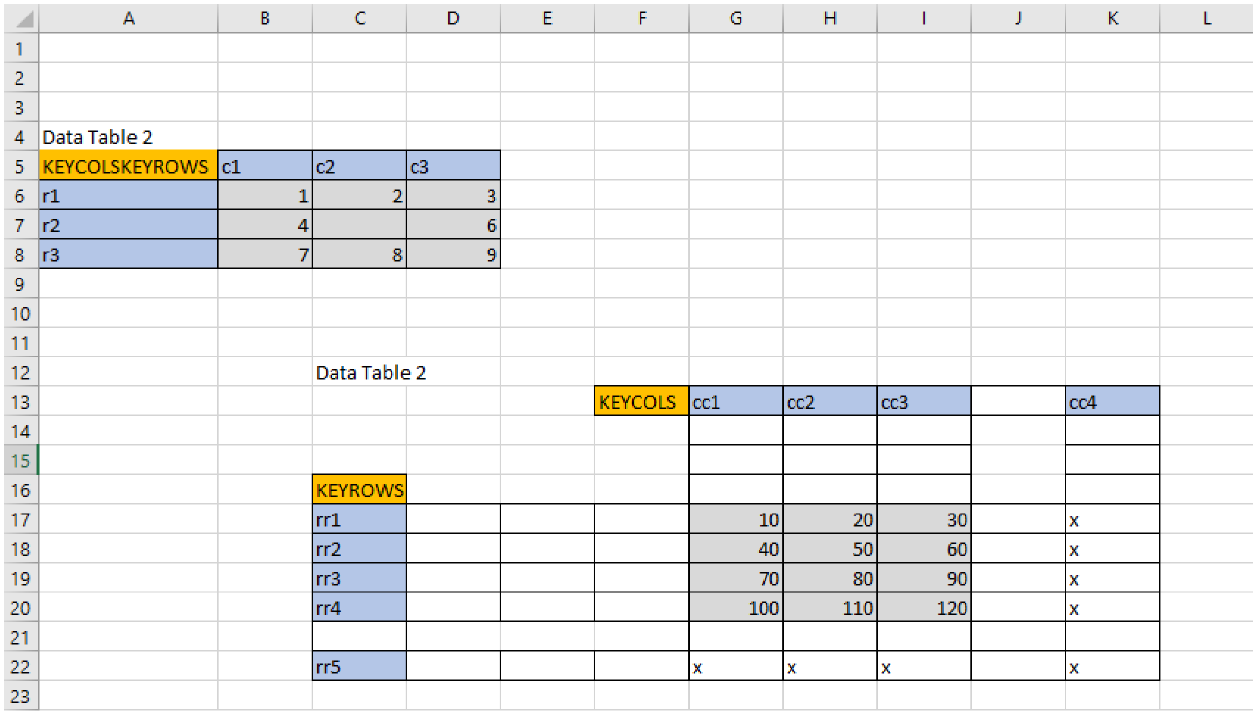
2.4.1 Reading Semi Structured data

- Question posted on stackoverflow
- Problem: read groups of 2D data from a single sheet that can begin at any row/col and has any number of rows/columns per data group, see figure below.
- Solution: note that `ssd` function takes any key-word argument as your KEYROWS/KEYCOLS flag and multiple tables are read the same way as you would read a book. Top left-to-right, then down.

```
import pylightxl
db = pylightxl.readxl('Book1.xlsx')

# request a semi-structured data (ssd) output
ssd = db.ws('Sheet1').ssd(keycols="KEYCOLS", keyrows="KEYROWS")

ssd[0]
>>> {'keyrows': ['r1', 'r2', 'r3'], 'keycols': ['c1', 'c2', 'c3'], 'data': [[1, 2, 3],
↪ [4, '', 6], [7, 8, 9]]}
ssd[1]
>>> {'keyrows': ['rr1', 'rr2', 'rr3', 'rr4'], 'keycols': ['cc1', 'cc2', 'cc3'], 'data
↪': [[10, 20, 30], [40, 50, 60], [70, 80, 90], [100, 110, 120]]}
```



	A	B	C	D	E	F	G	H	I	J	K	L
1												
2												
3												
4	Data Table 2											
5	KEYCOLSKEYROWS	c1	c2	c3								
6	r1	1	2	3								
7	r2	4		6								
8	r3	7	8	9								
9												
10												
11												
12			Data Table 2									
13						KEYCOLS	cc1	cc2	cc3		cc4	
14												
15												
16			KEYROWS									
17			rr1				10	20	30		x	
18			rr2				40	50	60		x	
19			rr3				70	80	90		x	
20			rr4				100	110	120		x	
21												
22			rr5				x	x	x		x	
23												

2.5 Revision Log

2.5.1 pypi version 1.61

- bug-fix: occasionally a <*definedName*> tag would case pylightxl to add duplicate of the same worksheet, see issue #75
- update: updated date handling (code cleanup)
- added feature: added python2 compatible typing to the library
- added feature: added *io.StringIO* support to *readcsv*

2.5.2 pypi version 1.60

- added feature: ability to update NamedRanges *wb.update_nr(name, val)*, see issue #72
- added feature: ability to find where a NamedRange is *wb.nr_loc(name)*
- added feature: ability to fill a range with a single value: *wb.ws('Sheet1').update_range(address='A1:B3', val=10)*
- update: NamedRanges now add the worksheets if they are not already in the workbook. Note that using *readxl* with worksheet names specified will also ignore NamedRanges from being read in from the sheet that are not read in.
- update: updated quickstart docs with the new feature demo scripts

2.5.3 pypi version 1.59

- bug fix: error in printing formulas that were read in as None type, see issue #59

- bug fix: added custom datetime and time style handling, see issue #36

2.5.4 pypi version 1.58

- improvement: added support for non-standard sheet ids, see issue #55
- improvement: added support for general IO file type inputs, see issue #57

2.5.5 pypi version 1.57

- improvement: added support for non-standard sheet ids (created by 3rd party tools), see issue #53
- improvement: added support for writing to existing sheets that contain excel customization, see issue #54

2.5.6 pypi version 1.56

- imporvement: added support for non-standard excel file xml tags, see issue #44
- bug fix: fixed keyrow bug, see issue #47
- bug fix: addressed csv writing issue related to cells that contain 'n' that previous started a new row. New version replaces 'n' with '', see issue #49
- bug fix: newly written workbooks written by pylightxl could not create new worksheets within excel after opening. The fix was to removed sheetView xml tag, see issue #50
- improvement: added encoding='utf-8' to write altworksheets to support chinese encoding error, see issue #51

2.5.7 pypi version 1.55

- added comment parsing, see issue #41
- DEPRECATION WARNING: all indexing method that use "formula" as an argument will be replaced with "output" in future version. Please update your codebase to use "output" instead of "formula". This was done to simplify indexing the value (`output='v'`), the formula (`output='f'`) or the comment (`output='c'`).
- added file stream reading for `readxl` that now supports `with block` for reading. See issue #25

2.5.8 pypi version 1.54

- added handling for datetime parsing

2.5.9 pypi version 1.53

- bug fix: writing to existing file previously would only write to the current working directory, it now can handle subdirs. In addition inadvertently discovered a bug in python source code ElementTree.iterparse where `source` passed as a string was not closing the file properly. We submitted a issue to python issue tracker.

2.5.10 pypi version 1.52

- updated reading error'ed cells "#N/A"
- updated workbook indexing bug from program generated workbooks that did not index from 1

2.5.11 pypi version 1.51

- license update within setup.py

2.5.12 pypi version 1.50

- hot-fix: added python2 support for encoding with cgi instead of html

2.5.13 pypi version 1.49

- bug-fix: updated encoding for string cells that contained xml-like data (ex: cell A1 "<cell content>")

2.5.14 pypi version 1.48

- add feature to `writecsv` to be able to handle `pathlib` object and `io.StreamIO` object
- refactored readxl to remove regex, now readxl is all cElementTree
- refactored readxl/writexl to able to handle excel files written by openpyxl that is generated differently than how excel write files.

2.5.15 pypi version 1.47

- added new function: `db.nr('table1')` returns the contents of named range "table1"
- added new function: `db.ws('Sheet1').range('A1:C3')` that returns the contents of a range it also has the ability to return the formulas of the range
- updated `db.ws('Sheet1').row()` and `db.ws('Sheet1').col()` to take in a new argument `formual` that returns the formulas of a row or col
- bugfix: write to existing without named ranges was throwing a "repair" error. Fixed typo on xml for it and added unit tests to capture it
- added new function: `xl.readcsv(fn, delimiter, ws)` to read csv files and create a pylightxl db out of it (type converted)
- added new function: `xl.writecsv(db, fn, ws, delimiter)` to write out a pylightxl worksheet as a csv

2.5.16 pypi version 1.46

- bug fix: added ability to input an empty string into the cell update functions (previously entering val='') threw and error

2.5.17 pypi version 1.45

- added support for cell values that have multiple formats within a single cell. previous versions did not support this functionality since it is logged differently in sharedString.xml
- added support for updating formulas and viewing them:
  - view formula: `db.ws('Sheet1').address('A1', formula=True)`

  - edit formula: `db.ws('Sheet1').update_address('A1', val='=A1+10')`

- updated the following function arguments to drive commonality:
  - was: `readxl(fn, sheetnames)` new: `readxl(fn, ws)`
  - was: `writexl(db, path)` new: `writexl(db, fn)`
  - was: `db.ws(sheetname)` new: `db.ws(ws)`
  - was: `db.add_ws(sheetname, data)` new: `db.add_ws(ws, data)`
- added new feature to be able to read-in NamedRanges, store it in the Database, update it, remove it, and write it. NamedRanges were integrated with existing function to handle semi-structured-data
  - `db.add_nr(name'range1', ws='sheet1', address='A1:C2')`
  - `db.remove_nr(name='range1')`
  - `db.nr_names`
- add feature to remove worksheet: `db.remove_ws(ws='Sheet1')`
- add feature to rename worksheet: `db.rename_ws(old='sh1', new='sh2')`
- added a cleanup function upon writing to delete _pylightxl_ temp folder in case an error left them
- added feature to write to file that is open by excel by appending a "**new_**" tag to the file name and a warning message that file is opened by excel so a file was saved as "**new_**" + filename

2.5.18 pypi version 1.44

- bug fix: accounted for num2letter roll-over issue
- new feature: added a pylightxl native function for handling semi-structured data

2.5.19 pypi version 1.43

- bug fix: accounted for reading error'ed out cell "#N/A"
- bug fix: accounted for bool TRUE/FALSE cell values not registering on readxl
- bug fix: accounted for edge case that was prematurely splitting cell tags <c r /> by formula closing bracket <f />
- bug fix: accounted for cell address roll-over

2.5.20 pypi version 1.42

- added support for pathlib file reading
- bug fix: previous version did not handle merged cells properly
- bug fix: database updates did not update maxcol maxrow if new data addition was larger than the initial dataset
- bug fix: writexl that use linefeeds did not read in properly into readxl (fixed regex)
- bug fix: writexl filepath issues

2.5.21 pypi version 1.41

- new-feature: write new excel file from pylightxl.Database
- new-feature: write to existing excel file from pylightxl.Database
- new-feature: db.update_index(row, col, val) for user defined cell values
- new-feature: db.update_address(address, val) for user defined cell values
- bug fix for reading user defined sheets
- bug fix for mis-alignment of reading user defined sheets and xml files

2.5.22 pypi version 1.3

- new-feature: add the ability to call rows/cols via key-value ex: `db.ws('Sheet1').keycol('my column header')` will return the entire column that has 'my column header' in row 1
- fixed-bug: fixed leading/trailing spaced cell text values that are marked `<t xml:space="preserve">` in the sharedString.xml

2.5.23 pypi version 1.2

- fixed-bug: fixed Sheet number to custom Sheet name matching for 10+ sheets that were previously only sorting alphabetical which resulted with sorting: Sheet1, Sheet10, Sheet11, Sheet2… and so on.

2.5.24 pypi version 1.1

- initial release

2.6 License

2.7 Contributor Code of Conduct

Pylightxl as with many other python open source projects practices an open and welcoming community for users and contributors. The following is our minimum expectation:

- A diverse community is a great community, and everyone is welcomed here, no matter the age, race, gender or background.
- Practice upmost respect for our users and maintainers. Maintainers answering issues on github or stackoverflow should always start by thanking the individual for considering using the tool, then begin to help them. Users, please be patient with our developers we try to test as much we can, but there will always be edge cases - just know that we are here to help you!

CHAPTER 3

Support Content Creator

If you have enjoy using this library please consider supporting it by one or more of the following ways:

- Star us on github! Github
- Sponsor via Tidelift
- Sponsor via Patreon

Index

A

`add_nr()` (*pylightxl.pylightxl.Database method*), 12
`add_ws()` (*pylightxl.pylightxl.Database method*), 12
`address()` (*pylightxl.pylightxl.Worksheet method*), 14

C

`col()` (*pylightxl.pylightxl.Worksheet method*), 14
`cols` (*pylightxl.pylightxl.Worksheet attribute*), 14

D

`Database` (*class in pylightxl.pylightxl*), 12

I

`index()` (*pylightxl.pylightxl.Worksheet method*), 14

K

`keycol()` (*pylightxl.pylightxl.Worksheet method*), 14
`keyrow()` (*pylightxl.pylightxl.Worksheet method*), 15

N

`nr()` (*pylightxl.pylightxl.Database method*), 12
`nr_loc()` (*pylightxl.pylightxl.Database method*), 13
`nr_names` (*pylightxl.pylightxl.Database attribute*), 13

R

`range()` (*pylightxl.pylightxl.Worksheet method*), 15
`readxl()` (*in module pylightxl.pylightxl*), 11
`remove_nr()` (*pylightxl.pylightxl.Database method*), 13
`remove_ws()` (*pylightxl.pylightxl.Database method*), 13
`rename_ws()` (*pylightxl.pylightxl.Database method*), 13
`row()` (*pylightxl.pylightxl.Worksheet method*), 15
`rows` (*pylightxl.pylightxl.Worksheet attribute*), 15

S

`set_emptycell()` (*pylightxl.pylightxl.Database method*), 13
`set_emptycell()` (*pylightxl.pylightxl.Worksheet method*), 15
`size` (*pylightxl.pylightxl.Worksheet attribute*), 16
`ssd()` (*pylightxl.pylightxl.Worksheet method*), 16

U

`update_address()` (*pylightxl.pylightxl.Worksheet method*), 16
`update_index()` (*pylightxl.pylightxl.Worksheet method*), 16
`update_nr()` (*pylightxl.pylightxl.Database method*), 13
`update_range()` (*pylightxl.pylightxl.Worksheet method*), 16
`utility_address2index()` (*in module pylightxl.pylightxl*), 16
`utility_columnletter2num()` (*in module pylightxl.pylightxl*), 17
`utility_index2address()` (*in module pylightxl.pylightxl*), 17
`utility_num2columnletters()` (*in module pylightxl.pylightxl*), 17

W

`Worksheet` (*class in pylightxl.pylightxl*), 14
`writexl()` (*in module pylightxl.pylightxl*), 12
`ws()` (*pylightxl.pylightxl.Database method*), 13
`ws_names` (*pylightxl.pylightxl.Database attribute*), 13