pyexcel

Release 0.5.15

Nov 07, 2019

Contents

1 Introduction **3**

2 Support the project **5**
- 2.1 Installation . . . 5
- 2.2 Usage . . . 7
- 2.3 Design . . . 7
- 2.4 New tutorial . . . 15
- 2.5 Old tutorial . . . 44
- 2.6 Cook book . . . 75
- 2.7 Real world cases . . . 84
- 2.8 API documentation . . . 87
- 2.9 Developer's guide . . . 89
- 2.10 Change log . . . 94

3 Indices and tables **109**

Bibliography **111**

Author C.W.

Source code http://github.com/pyexcel/pyexcel.git

Issues http://github.com/pyexcel/pyexcel/issues

License New BSD License

Released 0.5.15

Generated Nov 07, 2019

CHAPTER 1

Introduction

pyexcel provides **one** application programming interface to read, manipulate and write data in various excel formats. This library makes information processing involving excel files an enjoyable task. The data in excel files can be turned into *array or dict* with minimal code and vice versa. This library focuses on data processing using excel files as storage media hence fonts, colors and charts were not and will not be considered.

The idea originated from the common usability problem: when an excel file driven web application is delivered for non-developer users (ie: team assistant, human resource administrator etc). The fact is that not everyone knows (or cares) about the differences between various excel formats: csv, xls, xlsx are all the same to them. Instead of training those users about file formats, this library helps web developers to handle most of the excel file formats by providing a common programming interface. To add a specific excel file format type to you application, all you need is to install an extra pyexcel plugin. Hence no code changes to your application and no issues with excel file formats any more. Looking at the community, this library and its associated ones try to become a small and easy to install alternative to Pandas.

CHAPTER 2

Support the project

If your company has embedded pyexcel and its components into a revenue generating product, please support me on patreon or bounty source to maintain the project and develop it further.

If you are an individual, you are welcome to support me too and for however long you feel like. As my backer, you will receive early access to pyexcel related contents.

And your issues will get prioritized if you would like to become my patreon as *pyexcel pro user*.

With your financial support, I will be able to invest a little bit more time in coding, documentation and writing interesting posts.

2.1 Installation

You can install pyexcel via pip:

```
$ pip install pyexcel
```

or clone it and install it:

```
$ git clone https://github.com/pyexcel/pyexcel.git
$ cd pyexcel
$ python setup.py install
```

For individual excel file formats, please install them as you wish:

Table 1: A list of file formats supported by external plugins

Package name	Supported file formats	Dependencies	Python versions
pyexcel-io	csv, csvz [#f1]_, tsv, tsvz [#f2]_		2.6, 2.7, 3.3, 3.4, 3.5, 3.6 pypy
pyexcel-xls	xls, xlsx(read only), xlsm(read only)	xlrd, xlwt	same as above
pyexcel-xlsx	xlsx	openpyxl	same as above
pyexcel-ods3	ods	pyexcel-ezodf, lxml	2.6, 2.7, 3.3, 3.4 3.5, 3.6
pyexcel-ods	ods	odfpy	same as above

Table 2: Dedicated file reader and writers

Package name	Supported file formats	Dependencies	Python versions
pyexcel-xlsxw	xlsx(write only)	XlsxWriter	Python 2 and 3
pyexcel-xlsxr	xlsx(read only)	lxml	same as above
pyexcel-xlsbr	xlsx(read only)	pyxlsb	same as above
pyexcel-odsr	read only for ods, fods	lxml	same as above
pyexcel-odsw	write only for ods	loxun	same as above
pyexcel-htmlr	html(read only)	lxml,html5lib	same as above
pyexcel-pdfr	pdf(read only)	pdftables	Python 2 only.

Table 3: Other data renderers

Package name	Supported file formats	Dependencies	Python versions
pyexcel-text	write only:rst, mediawiki, html, latex, grid, pipe, orgtbl, plain simple read only: ndjson r/w: json	tabulate	2.6, 2.7, 3.3, 3.4 3.5, 3.6, pypy
pyexcel-handsontable	handsontable in html	handsontable	same as above
pyexcel-pygal	svg chart	pygal	2.7, 3.3, 3.4, 3.5 3.6, pypy
pyexcel-sortable	sortable table in html	csvtotable	same as above
pyexcel-gantt	gantt chart in html	frappe-gantt	except pypy, same as above

In order to manage the list of plugins installed, you need to use pip to add or remove a plugin. When you use virtualenv, you can have different plugins per virtual environment. In the situation where you have multiple plugins that does the same thing in your environment, you need to tell pyexcel which plugin to use per function call. For example, pyexcel-ods and pyexcel-odsr, and you want to get_array to use pyexcel-odsr. You need to append get_array(..., library='pyexcel-odsr').

For compatibility tables of pyexcel-io plugins, please click here

Table 4: Plugin compatibility table

pyexcel	pyexcel-io	pyexcel-text	pyexcel-handsontable	pyexcel-pygal	pyexcel-gantt
0.5.15+	0.5.19+	0.2.6+	0.0.1+	0.0.1	0.0.1
0.5.14	0.5.18	0.2.6+	0.0.1+	0.0.1	0.0.1
0.5.10+	0.5.11+	0.2.6+	0.0.1+	0.0.1	0.0.1
0.5.9.1+	0.5.9.1+	0.2.6+	0.0.1	0.0.1	0.0.1
0.5.4+	0.5.1+	0.2.6+	0.0.1	0.0.1	0.0.1
0.5.0+	0.4.0+	0.2.6+	0.0.1	0.0.1	0.0.1
0.4.0+	0.3.0+	0.2.5			

Table 5: a list of support file formats

file format	definition
csv	comma separated values
tsv	tab separated values
csvz	a zip file that contains one or many csv files
tsvz	a zip file that contains one or many tsv files
xls	a spreadsheet file format created by MS-Excel 97-2003 [#f1]_
xlsx	MS-Excel Extensions to the Office Open XML SpreadsheetML File Format. [#f2]_
xlsm	an MS-Excel Macro-Enabled Workbook file
ods	open document spreadsheet
fods	flat open document spreadsheet
json	java script object notation
html	html table of the data structure
simple	simple presentation
rst	rStructured Text presentation of the data
mediawiki	media wiki table

2.2 Usage

Suppose you want to process the following excel data :

Here are the example usages:

```
>>> import pyexcel as pe
>>> records = pe.iget_records(file_name="your_file.xls")
>>> for record in records:
...     print("%s is aged at %d" % (record['Name'], record['Age']))
Adam is aged at 28
Beatrice is aged at 29
Ceri is aged at 30
Dean is aged at 26
>>> pe.free_resources()
```

2.3 Design

2.3.1 Introduction

This section introduces Excel data models, its representing data structures and provides an overview of formatting, transformation, manipulation supported by **pyexcel**.

Data models and data structures

When dealing with excel files, **pyexcel** pay attention to three primary objects: **cell**, **sheet** and **book**.

A book contains one or more sheets and a sheet is consisted of a sheet name and a two dimensional array of cells. Although a sheet can contain charts and a cell can have formula, styling properties, this library ignores them and only pays attention to the data in the cell and its data type. So, in the context of this library, the definition of those three concepts are:

concept	definition	pyexcel data model
a cell	is a data unit	a Python data type
a sheet	is a named two dimensional array of data units	`Sheet`
a book	is a dictionary of two dimensional array of data units.	`Book`

Data source

A data source is a storage format of structured data. The most popular data source is an excel file. Libre Office/Microsoft Excel can easily be used to generate an excel file of your desired format. Besides a physical file, this library recognizes three additional types of source:

1. Excel files in computer memory. For example: when a file is uploaded to a Python server for information processing. If it is relatively small, it can be stored in memory.
2. Database tables. For example: a client would like to have a snapshot of some database table in an excel file and asks it to be sent to him.
3. Python structures. For example: a developer may have scraped a site and have stored data in Python array or dictionary. He may want to save this information as a file.

Reading from - and writing to - a data source is modelled as parsers and renderers in pyexcel. Excel data sources and database sources support read and write. Other data sources may only support read only, or write only methods.

Here is a list of data sources:

Data source	Read and write properties
Array	Read and write
Dictionary	Same as above
Records	Same as above
Excel files	Same as above
Excel files in memory	Same as above
Excel files on the web	Read only
Django models	Read and write
SQL models	Read and write
Database querysets	Read only
Textual sources	Write only

Data format

This library and its plugins support most of the frequently used excel file formats.

file format	definition
csv	comma separated values
tsv	tab separated values
csvz	a zip file that contains one or many csv files
tsvz	a zip file that contains one or many tsv files
xls	a spreadsheet file format created by MS-Excel 97-2003[1]
xlsx	MS-Excel Extensions to the Office Open XML SpreadsheetML File Format.[2]
xlsm	an MS-Excel Macro-Enabled Workbook file
ods	open document spreadsheet
json	java script object notation
html	html table of the data structure
simple	*simple* presentation
rst	rStructured Text presentation of the data
mediawiki	media wiki table

See also *A list of file formats supported by external plugins*.

Data transformation

Often a developer would like to have excel data imported into a Python data structure. This library supports the *conversions from* previous three data source to the following list of data structures, and *vice versa*.

Table 6: A list of supported data structures

Pesudo name	Python name	Related model
two dimensional array	a list of lists	`pyexcel.Sheet`
a dictionary of key value pair	a dictionary	`pyexcel.Sheet`
a dictionary of one dimensional arrays	a dictionary of lists	`pyexcel.Sheet`
a list of dictionaries	a list of dictionaries	`pyexcel.Sheet`
a dictionary of two dimensional arrays	a dictionary of lists of lists	`pyexcel.Book`

Data manipulation

The main operation on a cell involves *cell access*, *formatting* and *cleansing*. The main operation on a sheet involves group access to a row or a column; data filtering; and data transformation. The main operation in a book is obtain access to individual sheets.

Data transcoding

For various reasons the data in one format needs to be transcoded into another. This library provides a transcoding tunnel for data transcoding between supported file formats.

Data visualization

Via `pyexel.renderer.AbstractRenderer` interface, data visualization is made possible. **pyexcel-chart** is the interface plugin to formalize this effort. **pyexcel-pygal** is the first plugin to provide bar, pie, histogram charts and more.

[1] quoted from whatis.com. Technical details can be found at MSDN XLS
[2] xlsx is used by MS-Excel 2007, more information can be found at MSDN XLSX

Examples of supported data structure

Here is a list of examples:

```
>>> import pyexcel as p
>>> two_dimensional_list = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
>>> p.get_sheet(array=two_dimensional_list)
pyexcel_sheet1:
+---+----+----+----+
| 1 | 2  | 3  | 4  |
+---+----+----+----+
| 5 | 6  | 7  | 8  |
+---+----+----+----+
| 9 | 10 | 11 | 12 |
+---+----+----+----+
>>> a_dictionary_of_key_value_pair = {
...     "IE": 0.2,
...     "Firefox": 0.3
... }
>>> p.get_sheet(adict=a_dictionary_of_key_value_pair)
pyexcel_sheet1:
+---------+-----+
| Firefox | IE  |
+---------+-----+
| 0.3     | 0.2 |
+---------+-----+
>>> a_dictionary_of_one_dimensional_arrays = {
...      "Column 1": [1, 2, 3, 4],
...      "Column 2": [5, 6, 7, 8],
...      "Column 3": [9, 10, 11, 12],
... }
>>> p.get_sheet(adict=a_dictionary_of_one_dimensional_arrays)
pyexcel_sheet1:
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+----------+----------+----------+
| 1        | 5        | 9        |
+----------+----------+----------+
| 2        | 6        | 10       |
+----------+----------+----------+
| 3        | 7        | 11       |
+----------+----------+----------+
| 4        | 8        | 12       |
+----------+----------+----------+
>>> a_list_of_dictionaries = [
...      {
...          "Name": 'Adam',
...          "Age": 28
...      },
...      {
...          "Name": 'Beatrice',
...          "Age": 29
...      },
...      {
```

(continues on next page)

(continued from previous page)

```
...         "Name": 'Ceri',
...         "Age": 30
...     },
...     {
...         "Name": 'Dean',
...         "Age": 26
...     }
... ]
>>> p.get_sheet(records=a_list_of_dictionaries)
pyexcel_sheet1:
+-----+----------+
| Age | Name     |
+-----+----------+
| 28  | Adam     |
+-----+----------+
| 29  | Beatrice |
+-----+----------+
| 30  | Ceri     |
+-----+----------+
| 26  | Dean     |
+-----+----------+
>>> a_dictionary_of_two_dimensional_arrays = {
...     'Sheet 1':
...         [
...             [1.0, 2.0, 3.0],
...             [4.0, 5.0, 6.0],
...             [7.0, 8.0, 9.0]
...         ],
...     'Sheet 2':
...         [
...             ['X', 'Y', 'Z'],
...             [1.0, 2.0, 3.0],
...             [4.0, 5.0, 6.0]
...         ],
...     'Sheet 3':
...         [
...             ['O', 'P', 'Q'],
...             [3.0, 2.0, 1.0],
...             [4.0, 3.0, 2.0]
...         ]
...  }
>>> p.get_book(bookdict=a_dictionary_of_two_dimensional_arrays)
Sheet 1:
+-----+-----+-----+
| 1.0 | 2.0 | 3.0 |
+-----+-----+-----+
| 4.0 | 5.0 | 6.0 |
+-----+-----+-----+
| 7.0 | 8.0 | 9.0 |
+-----+-----+-----+
Sheet 2:
+-----+-----+-----+
| X   | Y   | Z   |
+-----+-----+-----+
| 1.0 | 2.0 | 3.0 |
+-----+-----+-----+
| 4.0 | 5.0 | 6.0 |
```

(continues on next page)

(continued from previous page)

```
+-----+-----+-----+
Sheet 3:
+-----+-----+-----+
| O   | P   | Q   |
+-----+-----+-----+
| 3.0 | 2.0 | 1.0 |
+-----+-----+-----+
| 4.0 | 3.0 | 2.0 |
+-----+-----+-----+
```

2.3.2 Signature functions

Import data into Python

This library provides one application programming interface to read data from one of the following data sources:

- physical file
- memory file
- SQLAlchemy table
- Django Model
- Python data structures: dictionary, records and array

and to transform them into one of the following data structures:

- two dimensional array
- a dictionary of one dimensional arrays
- a list of dictionaries
- a dictionary of two dimensional arrays
- a `Sheet`
- a `Book`

Four data access functions

Python data can be handled well using lists, dictionaries and various mixture of both. This library provides four module level functions to help you obtain excel data in these data structures. Please refer to "A list of module level functions", the first three functions operates on any one sheet from an excel book and the fourth one returns all data in all sheets in an excel book.

Table 7: A list of module level functions

Functions	Name	Python name
`get_array()`	two dimensional array	a list of lists
`get_dict()`	a dictionary of one dimensional arrays	an ordered dictionary of lists
`get_records()`	a list of dictionaries	a list of dictionaries
`get_book_dict()`	a dictionary of two dimensional arrays	a dictionary of lists of lists

See also:

- get_an_array_from_an_excel_sheet

- *How to get a dictionary from an excel sheet*
- *How to obtain records from an excel sheet*
- *How to obtain a dictionary from a multiple sheet book*

The following two variants of the data access function use generator and should work well with big data files

Table 8: A list of variant functions

Functions	Name	Python name
`iget_array()`	**a memory efficient two dimensional** array	a generator of a list of lists
`iget_records()`	a memory efficient list list of dictionaries	a generator of a list of dictionaries

However, you will need to call `free_resource()` to make sure file handles are closed.

Two pyexcel functions

In cases where the excel data needs custom manipulations, a pyexcel user got a few choices: one is to use `Sheet` and `Book`, the other is to look for more sophisticated ones:

- Pandas, for numerical analysis
- Do-it-yourself

Functions	Returns
`get_sheet()`	`Sheet`
`get_book()`	`Book`

For all six functions, you can pass on the same command parameters while the return value is what the function says.

Export data from Python

This library provides one application programming interface to transform them into one of the data structures:

- two dimensional array
- a (ordered) dictionary of one dimensional arrays
- a list of dictionaries
- a dictionary of two dimensional arrays
- a `Sheet`
- a `Book`

and write to one of the following data sources:

- physical file
- memory file
- SQLAlchemy table
- Django Model

- Python data structures: dictionary, records and array

Here are the two functions:

Functions	Description
`save_as()`	Works well with single sheet file
`isave_as()`	Works well with big data files
`save_book_as()`	**Works with multiple sheet file** and big data files
`isave_book_as()`	**Works with multiple sheet file** and big data files

If you would only use these two functions to do format transcoding, you may enjoy a speed boost using `isave_as()` and `isave_book_as()`, because they use *yield* keyword and minimize memory footprint. However, you will need to call `free_resource()` to make sure file handles are closed. And `save_as()` and `save_book_as()` reads all data into memory and **will make all rows the same width**.

See also:

- *How to save an python array as an excel file*
- *How to save a dictionary of two dimensional array as an excel file*
- *How to save an python array as a csv file with special delimiter*

Data transportation/transcoding

This library is capable of transporting your data between any of the following data sources:

- physical file
- memory file
- SQLAlchemy table
- Django Model
- Python data structures: dictionary, records and array

See also:

- *How to import an excel sheet to a database using SQLAlchemy*
- *How to open an xls file and save it as xlsx*
- *How to open an xls file and save it as csv*

2.3.3 Architecture

pyexcel uses loosely couple plugins to fullfil the promise to access various file formats. **lml** is the plugin management library that provide the specialized support for the loose coupling.

What is loose coupling?

The components of **pyexcel** is designed as building blocks. For your project, you can cherry-pick the file format support without affecting the core functionality of pyexcel. Each plugin will bring in additional dependences. For example, if you choose pyexcel-xls, xlrd and xlwt will be brought in as 2nd level depndencies.

Looking at the following architectural diagram, pyexcel hosts plugin interfaces for data source, data renderer and data parser. pyexcel-pygal, pyexcel-matplotlib, and pyexcel-handsontable extend pyexcel using data renderer interface. pyexcel-io package takes away the responsibilities to interface with excel libraries, for example: xlrd, openpyxl, ezodf.

As in *A list of file formats supported by external plugins*, there are overlapping capabilities in reading and writing xlsx, ods files. Because each third parties express different personalities although they may read and write data in the same file format, you as the pyexcel is left to pick which suit your task best.

Dotted arrow means the package or module is loaded later.

2.4 New tutorial

2.4.1 One liners

This section shows you how to get data from your excel files and how to export data to excel files in **one line**

One liner to get data from the excel files

Get a list of dictionaries

Suppose you want to process the `following coffee data` (data source coffee chart on the center for science in the public interest):

Let's get a list of dictionary out from the xls file:

```
>>> records = p.get_records(file_name="your_file.xls")
```

And let's check what do we have:

```
>>> for record in records:
...     print("%s of %s has %s mg" % (
...         record['Serving Size'],
...         record['Coffees'],
...         record['Caffeine (mg)']))
venti(20 oz) of Starbucks Coffee Blonde Roast has 475 mg
large(20 oz.) of Dunkin' Donuts Coffee with Turbo Shot has 398 mg
grande(16 oz.) of Starbucks Coffee Pike Place Roast has 310 mg
regular(16 oz.) of Panera Coffee Light Roast has 300 mg
```

Get two dimensional array

Instead, what if you have to use `pyexcel.get_array()` to do the same:

```
>>> for row in p.get_array(file_name="your_file.xls", start_row=1):
...     print("%s of %s has %s mg" % (
...         row[1],
...         row[0],
...         row[2]))
venti(20 oz) of Starbucks Coffee Blonde Roast has 475 mg
large(20 oz.) of Dunkin' Donuts Coffee with Turbo Shot has 398 mg
```

(continues on next page)

(continued from previous page)

```
grande(16 oz.) of Starbucks Coffee Pike Place Roast has 310 mg
regular(16 oz.) of Panera Coffee Light Roast has 300 mg
```

where *start_row* skips the header row.

Get a dictionary

You can get a dictionary too:

Now let's get a dictionary out from the spreadsheet:

```
>>> my_dict = p.get_dict(file_name="your_file.xls", name_columns_by_row=0)
```

And check what do we have:

```
>>> from pyexcel._compact import OrderedDict
>>> isinstance(my_dict, OrderedDict)
True
>>> for key, values in my_dict.items():
...     print(key + " : " + ','.join([str(item) for item in values]))
Coffees : Starbucks Coffee Blonde Roast,Dunkin' Donuts Coffee with Turbo Shot,
↪Starbucks Coffee Pike Place Roast,Panera Coffee Light Roast
Serving Size : venti(20 oz),large(20 oz.),grande(16 oz.),regular(16 oz.)
Caffeine (mg) : 475,398,310,300
```

Please note that my_dict is an OrderedDict.

Get a dictionary of two dimensional array

Suppose you have a multiple sheet book as the following:

Here is the code to obtain those sheets as a single dictionary:

```
>>> book_dict = p.get_book_dict(file_name="book.xls")
```

And check::

```
>>> isinstance(book_dict, OrderedDict)
True
>>> import json
>>> for key, item in book_dict.items():
...     print(json.dumps({key: item}))
{"Sheet 1": [[1, 2, 3], [4, 5, 6], [7, 8, 9]]}
{"Sheet 2": [["X", "Y", "Z"], [1, 2, 3], [4, 5, 6]]}
{"Sheet 3": [["O", "P", "Q"], [3, 2, 1], [4, 3, 2]]}
```

Data export in one line

Export an array

Suppose you have the following array:

```
>>> data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

And here is the code to save it as an excel file

```
>>> p.save_as(array=data, dest_file_name="example.xls")
```

Let's verify it:

```
>>> p.get_sheet(file_name="example.xls")
pyexcel_sheet1:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+
```

And here is the code to save it as a csv file

```
>>> p.save_as(array=data,
...            dest_file_name="example.csv",
...            dest_delimiter=':')
```

Let's verify it:

```
>>> with open("example.csv") as f:
...     for line in f.readlines():
...         print(line.rstrip())
...
1:2:3
4:5:6
7:8:9
```

Export a list of dictionaries

```
>>> records = [
...     {"year": 1903, "country": "Germany", "speed": "206.7km/h"},
...     {"year": 1964, "country": "Japan", "speed": "210km/h"},
...     {"year": 2008, "country": "China", "speed": "350km/h"}
... ]
>>> p.save_as(records=records, dest_file_name='high_speed_rail.xls')
```

Export a dictionary of single key value pair

```
>>> henley_on_thames_facts = {
...     "area": "5.58 square meters",
...     "population": "11,619",
...     "civial parish": "Henley-on-Thames",
...     "latitude": "51.536",
...     "longitude": "-0.898"
... }
>>> p.save_as(adict=henley_on_thames_facts, dest_file_name='henley.xlsx')
```

Export a dictionary of single dimensonal array

```
>>> ccs_insights = {
...     "year": ["2017", "2018", "2019", "2020", "2021"],
...     "smart phones": [1.53, 1.64, 1.74, 1.82, 1.90],
...     "feature phones": [0.46, 0.38, 0.30, 0.23, 0.17]
... }
>>> p.save_as(adict=ccs_insights, dest_file_name='ccs.csv')
```

Export a dictionary of two dimensional array as a book

Suppose you want to save the below dictionary to an excel file

```
>>> a_dictionary_of_two_dimensional_arrays = {
...      'Sheet 1':
...          [
...              [1.0, 2.0, 3.0],
...              [4.0, 5.0, 6.0],
...              [7.0, 8.0, 9.0]
...          ],
...      'Sheet 2':
...          [
...              ['X', 'Y', 'Z'],
...              [1.0, 2.0, 3.0],
...              [4.0, 5.0, 6.0]
...          ],
...      'Sheet 3':
...          [
...              ['O', 'P', 'Q'],
...              [3.0, 2.0, 1.0],
...              [4.0, 3.0, 2.0]
...          ]
...  }
```

Here is the code:

```
>>> p.save_book_as(
...    bookdict=a_dictionary_of_two_dimensional_arrays,
...    dest_file_name="book.xls"
... )
```

If you want to preserve the order of sheets in your dictionary, you have to pass on an ordered dictionary to the function itself. For example:

```
>>> data = OrderedDict()
>>> data.update({"Sheet 2": a_dictionary_of_two_dimensional_arrays['Sheet 2']})
>>> data.update({"Sheet 1": a_dictionary_of_two_dimensional_arrays['Sheet 1']})
>>> data.update({"Sheet 3": a_dictionary_of_two_dimensional_arrays['Sheet 3']})
>>> p.save_book_as(bookdict=data, dest_file_name="book.xls")
```

Let's verify its order:

```
>>> book_dict = p.get_book_dict(file_name="book.xls")
>>> for key, item in book_dict.items():
...     print(json.dumps({key: item}))
```

(continues on next page)

(continued from previous page)

```
{"Sheet 2": [["X", "Y", "Z"], [1, 2, 3], [4, 5, 6]]}
{"Sheet 1": [[1, 2, 3], [4, 5, 6], [7, 8, 9]]}
{"Sheet 3": [["O", "P", "Q"], [3, 2, 1], [4, 3, 2]]}
```

Please notice that "Sheet 2" is the first item in the *book_dict*, meaning the order of sheets are preserved.

File format transcoding on one line

Note: Please note that the following file transcoding could be with zero line. Please install pyexcel-cli and you will do the transcode in one command. No need to open your editor, save the problem, then python run.

The following code does a simple file format transcoding from xls to csv:

```
>>> p.save_as(file_name="birth.xls", dest_file_name="birth.csv")
```

Again it is really simple. Let's verify what we have gotten:

```
>>> sheet = p.get_sheet(file_name="birth.csv")
>>> sheet
birth.csv:
+-------+--------+----------+
| name  | weight | birth    |
+-------+--------+----------+
| Adam  | 3.4    | 03/02/15 |
+-------+--------+----------+
| Smith | 4.2    | 12/11/14 |
+-------+--------+----------+
```

Note: Please note that csv(comma separate value) file is pure text file. Formula, charts, images and formatting in xls file will disappear no matter which transcoding tool you use. Hence, pyexcel is a quick alternative for this transcoding job.

Let use previous example and save it as xlsx instead

```
>>> p.save_as(file_name="birth.xls",
...           dest_file_name="birth.xlsx") # change the file extension
```

Again let's verify what we have gotten:

```
>>> sheet = p.get_sheet(file_name="birth.xlsx")
>>> sheet
pyexcel_sheet1:
+-------+--------+----------+
| name  | weight | birth    |
+-------+--------+----------+
| Adam  | 3.4    | 03/02/15 |
+-------+--------+----------+
| Smith | 4.2    | 12/11/14 |
+-------+--------+----------+
```

Excel book merge and split operation in one line

Merge all excel files in directory into a book where each file become a sheet

The following code will merge every excel files into one file, say "output.xls":

```
from pyexcel.cookbook import merge_all_to_a_book
import glob


merge_all_to_a_book(glob.glob("your_csv_directory\*.csv"), "output.xls")
```

You can mix and match with other excel formats: xls, xlsm and ods. For example, if you are sure you have only xls, xlsm, xlsx, ods and csv files in *your_excel_file_directory*, you can do the following:

```
from pyexcel.cookbook import merge_all_to_a_book
import glob


merge_all_to_a_book(glob.glob("your_excel_file_directory\*.*"), "output.xls")
```

Split a book into single sheet files

Suppose you have many sheets in a work book and you would like to separate each into a single sheet excel file. You can easily do this:

```
>>> from pyexcel.cookbook import split_a_book
>>> split_a_book("megabook.xls", "output.xls")
>>> import glob
>>> outputfiles = glob.glob("*_output.xls")
>>> for file in sorted(outputfiles):
...     print(file)
...
Sheet 1_output.xls
Sheet 2_output.xls
Sheet 3_output.xls
```

for the output file, you can specify any of the supported formats

Extract just one sheet from a book

Suppose you just want to extract one sheet from many sheets that exists in a work book and you would like to separate it into a single sheet excel file. You can easily do this:

```
>>> from pyexcel.cookbook import extract_a_sheet_from_a_book
>>> extract_a_sheet_from_a_book("megabook.xls", "Sheet 1", "output.xls")
>>> if os.path.exists("Sheet 1_output.xls"):
...     print("Sheet 1_output.xls exists")
...
Sheet 1_output.xls exists
```

for the output file, you can specify any of the supported formats

2.4.2 Stream APIs for big file : A set of two liners

This section shows you how to get data from your **BIG** excel files and how to export data to excel files in **two lines** at most.

Two liners for get data from big excel files

Get a list of dictionaries

Suppose you want to process the following coffee data:

Let's get a list of dictionary out from the xls file:

```
>>> records = p.iget_records(file_name="your_file.xls")
```

And let's check what do we have:

```
>>> for record in records:
...     print("%s of %s has %s mg" % (
...         record['Serving Size'],
...         record['Coffees'],
...         record['Caffeine (mg)']))
venti(20 oz) of Starbucks Coffee Blonde Roast has 475 mg
large(20 oz.) of Dunkin' Donuts Coffee with Turbo Shot has 398 mg
grande(16 oz.) of Starbucks Coffee Pike Place Roast has 310 mg
regular(16 oz.) of Panera Coffee Light Roast has 300 mg
```

Please do not forgot the second line:

```
>>> p.free_resources()
```

Get two dimensional array

Instead, what if you have to use `pyexcel.get_array()` to do the same:

```
>>> for row in p.iget_array(file_name="your_file.xls", start_row=1):
...     print("%s of %s has %s mg" % (
...         row[1],
...         row[0],
...         row[2]))
venti(20 oz) of Starbucks Coffee Blonde Roast has 475 mg
large(20 oz.) of Dunkin' Donuts Coffee with Turbo Shot has 398 mg
grande(16 oz.) of Starbucks Coffee Pike Place Roast has 310 mg
regular(16 oz.) of Panera Coffee Light Roast has 300 mg
```

Again, do not forgot the second line:

```
>>> p.free_resources()
```

where *start_row* skips the header row.

Data export in one liners

Export an array

Suppose you have the following array:

```
>>> data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

And here is the code to save it as an excel file

```
>>> p.isave_as(array=data, dest_file_name="example.xls")
```

But the following line is not required because the data source are not file sources:

```
>>> # p.free_resources()
```

Let's verify it:

```
>>> p.get_sheet(file_name="example.xls")
pyexcel_sheet1:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+
```

And here is the code to save it as a csv file

```
>>> p.isave_as(array=data,
...             dest_file_name="example.csv",
...             dest_delimiter=':')
```

Let's verify it:

```
>>> with open("example.csv") as f:
...     for line in f.readlines():
...         print(line.rstrip())
...
1:2:3
4:5:6
7:8:9
```

Export a list of dictionaries

```
>>> records = [
...     {"year": 1903, "country": "Germany", "speed": "206.7km/h"},
...     {"year": 1964, "country": "Japan", "speed": "210km/h"},
...     {"year": 2008, "country": "China", "speed": "350km/h"}
... ]
>>> p.isave_as(records=records, dest_file_name='high_speed_rail.xls')
```

Export a dictionary of single key value pair

```
>>> henley_on_thames_facts = {
...     "area": "5.58 square meters",
...     "population": "11,619",
...     "civial parish": "Henley-on-Thames",
...     "latitude": "51.536",
...     "longitude": "-0.898"
... }
>>> p.isave_as(adict=henley_on_thames_facts, dest_file_name='henley.xlsx')
```

Export a dictionary of single dimensonal array

```
>>> ccs_insights = {
...     "year": ["2017", "2018", "2019", "2020", "2021"],
...     "smart phones": [1.53, 1.64, 1.74, 1.82, 1.90],
...     "feature phones": [0.46, 0.38, 0.30, 0.23, 0.17]
... }
>>> p.isave_as(adict=ccs_insights, dest_file_name='ccs.csv')
>>> p.free_resources()
```

Export a dictionary of two dimensional array as a book

Suppose you want to save the below dictionary to an excel file

```
>>> a_dictionary_of_two_dimensional_arrays = {
...      'Sheet 1':
...          [
...              [1.0, 2.0, 3.0],
...              [4.0, 5.0, 6.0],
...              [7.0, 8.0, 9.0]
...          ],
...      'Sheet 2':
...          [
...              ['X', 'Y', 'Z'],
...              [1.0, 2.0, 3.0],
...              [4.0, 5.0, 6.0]
...          ],
...      'Sheet 3':
...          [
...              ['O', 'P', 'Q'],
...              [3.0, 2.0, 1.0],
...              [4.0, 3.0, 2.0]
...          ]
...  }
```

Here is the code:

```
>>> p.isave_book_as(
...    bookdict=a_dictionary_of_two_dimensional_arrays,
...    dest_file_name="book.xls"
... )
```

If you want to preserve the order of sheets in your dictionary, you have to pass on an ordered dictionary to the function itself. For example:

```
>>> from pyexcel._compact import OrderedDict
>>> data = OrderedDict()
>>> data.update({"Sheet 2": a_dictionary_of_two_dimensional_arrays['Sheet 2']})
>>> data.update({"Sheet 1": a_dictionary_of_two_dimensional_arrays['Sheet 1']})
>>> data.update({"Sheet 3": a_dictionary_of_two_dimensional_arrays['Sheet 3']})
>>> p.isave_book_as(bookdict=data, dest_file_name="book.xls")
>>> p.free_resources()
```

Let's verify its order:

```
>>> import json
>>> book_dict = p.get_book_dict(file_name="book.xls")
>>> for key, item in book_dict.items():
...     print(json.dumps({key: item}))
{"Sheet 2": [["X", "Y", "Z"], [1, 2, 3], [4, 5, 6]]}
{"Sheet 1": [[1, 2, 3], [4, 5, 6], [7, 8, 9]]}
{"Sheet 3": [["O", "P", "Q"], [3, 2, 1], [4, 3, 2]]}
```

Please notice that "Sheet 2" is the first item in the *book_dict*, meaning the order of sheets are preserved.

File format transcoding on one line

Note: Please note that the following file transcoding could be with zero line. Please install pyexcel-cli and you will do the transcode in one command. No need to open your editor, save the problem, then python run.

The following code does a simple file format transcoding from xls to csv:

```
>>> import pyexcel
>>> p.save_as(file_name="birth.xls", dest_file_name="birth.csv")
```

Again it is really simple. Let's verify what we have gotten:

```
>>> sheet = p.get_sheet(file_name="birth.csv")
>>> sheet
birth.csv:
+-------+--------+----------+
| name  | weight | birth    |
+-------+--------+----------+
| Adam  | 3.4    | 03/02/15 |
+-------+--------+----------+
| Smith | 4.2    | 12/11/14 |
+-------+--------+----------+
```

Note: Please note that csv(comma separate value) file is pure text file. Formula, charts, images and formatting in xls file will disappear no matter which transcoding tool you use. Hence, pyexcel is a quick alternative for this transcoding job.

Let use previous example and save it as xlsx instead

```
>>> import pyexcel
>>> p.isave_as(file_name="birth.xls",
...            dest_file_name="birth.xlsx") # change the file extension
```

Again let's verify what we have gotten:

```
>>> sheet = p.get_sheet(file_name="birth.xlsx")
>>> sheet
pyexcel_sheet1:
+-------+--------+----------+
| name  | weight | birth    |
+-------+--------+----------+
| Adam  | 3.4    | 03/02/15 |
+-------+--------+----------+
| Smith | 4.2    | 12/11/14 |
+-------+--------+----------+
```

2.4.3 For web developer

The following libraries are written to facilitate the daily import and export of excel data.

framework	plugin/middleware/extension
Flask	Flask-Excel
Django	django-excel
Pyramid	pyramid-excel

And you may make your own by using pyexcel-webio

Read any supported excel and respond its content in json

You can find a real world example in **examples/memoryfile/** directory: pyexcel_server.py. Here is the example snippet

```
1  def upload():
2      if request.method == 'POST' and 'excel' in request.files:
3          # handle file upload
4          filename = request.files['excel'].filename
5          extension = filename.split(".")[-1]
6          # Obtain the file extension and content
7          # pass a tuple instead of a file name
8          content = request.files['excel'].read()
9          if sys.version_info[0] > 2:
10             # in order to support python 3
11             # have to decode bytes to str
12             content = content.decode('utf-8')
13         sheet = pe.get_sheet(file_type=extension, file_content=content)
14         # then use it as usual
15         sheet.name_columns_by_row(0)
16         # respond with a json
17         return jsonify({"result": sheet.dict})
18     return render_template('upload.html')
```

request.files['excel'] in line 4 holds the file object. line 5 finds out the file extension. line 13 obtains a sheet instance. line 15 uses the first row as data header. line 17 sends the json representation of the excel file back to client browser.

Write to memory and respond to download

```
data = [
    [...],
    ...
]

@app.route('/download')
def download():
    sheet = pe.Sheet(data)
    output = make_response(sheet.csv)
    output.headers["Content-Disposition"] = "attachment; filename=export.csv"
    output.headers["Content-type"] = "text/csv"
    return output
```

make_response is a Flask utility to make a memory content as http response.

Note: You can find the corresponding source code at examples/memoryfile

2.4.4 Pyexcel data renderers

There exist a few data renderers for pyexcel data. This chapter will walk you through them.

View pyexcel data in ndjson and other formats

With pyexcel-text, you can get pyexcel data in newline delimited json, normal json and other formats.

View the pyexcel data in a browser

You can use pyexcel-handsontable to render your data.

Include excel data in your python documentation

sphinxcontrib-excel help you present your excel data in various formats inside your sphinx documentation.

Draw charts from your excel data

pyexcel-pygal helps you with all charting options and give you charts in svg format.

pyexcel-echarts draws 2D, 3D, geo charts from pyexcel data and has awesome animations too, but it is under development.

pyexcel-matplotlib helps you with scentific charts and is under developmement.

Gantt chart visualization for your excel data

'pyexcel-gantt'_ is a specialist renderer for gantt chart.

2.4.5 Sheet

Random access

To randomly access a cell of `Sheet` instance, two syntax are available:

```
sheet[row, column]
```

or:

```
sheet['A1']
```

The former syntax is handy when you know the row and column numbers. The latter syntax is introduced to help you convert the excel column header such as "AX" to integer numbers.

Suppose you have the following data, you can get value 5 by reader[2, 2].

Here is the example code showing how you can randomly access a cell:

```
>>> sheet = pyexcel.get_sheet(file_name="example.xls")
>>> sheet.content
+---------+---+---+---+
| Example | X | Y | Z |
+---------+---+---+---+
| a       | 1 | 2 | 3 |
+---------+---+---+---+
| b       | 4 | 5 | 6 |
+---------+---+---+---+
| c       | 7 | 8 | 9 |
+---------+---+---+---+
>>> print(sheet[2, 2])
5
>>> print(sheet["C3"])
5
>>> sheet[3, 3] = 10
>>> print(sheet[3, 3])
10
```

Note: In order to set a value to a cell, please use sheet[row_index, column_index] = new_value

Random access to rows and columns

Continue with previous excel file, you can access row and column separately:

```
>>> sheet.row[1]
['a', 1, 2, 3]
>>> sheet.column[2]
['Y', 2, 5, 8]
```

Use custom names instead of index Alternatively, it is possible to use the first row to refer to each columns:

```
>>> sheet.name_columns_by_row(0)
>>> print(sheet[1, "Y"])
5
>>> sheet[1, "Y"] = 100
>>> print(sheet[1, "Y"])
100
```

You have noticed the row index has been changed. It is because first row is taken as the column names, hence all rows after the first row are shifted. Now accessing the columns are changed too:

```
>>> sheet.column['Y']
[2, 100, 8]
```

Hence access the same cell, this statement also works:

```
>>> sheet.column['Y'][1]
100
```

Further more, it is possible to use first column to refer to each rows:

```
>>> sheet.name_rows_by_column(0)
```

To access the same cell, we can use this line:

```
>>> sheet.row["b"][1]
100
```

For the same reason, the row index has been reduced by 1. Since we have named columns and rows, it is possible to access the same cell like this:

```
>>> print(sheet["b", "Y"])
100
>>> sheet["b", "Y"] = 200
>>> print(sheet["b", "Y"])
200
```

Play with data

Suppose you have the following data in any of the supported excel formats again:

```
>>> sheet = pyexcel.get_sheet(file_name="example_series.xls",
...       name_columns_by_row=0)
```

You can get headers:

```
>>> print(list(sheet.colnames))
['Column 1', 'Column 2', 'Column 3']
```

You can use a utility function to get all in a dictionary:

```
>>> sheet.to_dict()
OrderedDict([('Column 1', [1, 4, 7]), ('Column 2', [2, 5, 8]), ('Column 3', [3, 6,␣
↪9])])
```

Maybe you want to get only the data without the column headers. You can call `rows()` instead:

```
>>> list(sheet.rows())
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

You can get data from the bottom to the top one by calling `rrows()`:

```
>>> list(sheet.rrows())
[[7, 8, 9], [4, 5, 6], [1, 2, 3]]
```

You might want the data arranged vertically. You can call `columns()`:

```
>>> list(sheet.columns())
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

You can get columns in reverse sequence as well by calling `rcolumns()`:

```
>>> list(sheet.rcolumns())
[[3, 6, 9], [2, 5, 8], [1, 4, 7]]
```

Do you want to flatten the data? You can get the content in one dimensional array. If you are interested in playing with one dimensional enumeration, you can check out these functions `enumerate()`, `reverse()`, `vertical()`, and `rvertical()`:

```
>>> list(sheet.enumerate())
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(sheet.reverse())
[9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> list(sheet.vertical())
[1, 4, 7, 2, 5, 8, 3, 6, 9]
>>> list(sheet.rvertical())
[9, 6, 3, 8, 5, 2, 7, 4, 1]
```

attributes

Attributes:

```
>>> import pyexcel
>>> content = "1,2,3\n3,4,5"
>>> sheet = pyexcel.get_sheet(file_type="csv", file_content=content)
>>> sheet.tsv
'1\t2\t3\r\n3\t4\t5\r\n'
>>> print(sheet.simple)
csv:
-  -  -
1  2  3
3  4  5
-  -  -
```

What's more, you could as well set value to an attribute, for example::

```
>>> import pyexcel
>>> content = "1,2,3\n3,4,5"
>>> sheet = pyexcel.Sheet()
>>> sheet.csv = content
>>> sheet.array
[[1, 2, 3], [3, 4, 5]]
```

You can get the direct access to underneath stream object. In some situation, it is desired:

```
>>> stream = sheet.stream.tsv
```

The returned stream object has tsv formatted content for reading.

What you could further do is to set a memory stream of any supported file format to a sheet. For example:

```
>>> another_sheet = pyexcel.Sheet()
>>> another_sheet.xls = sheet.xls
>>> another_sheet.content
+---+---+---+
```

(continues on next page)

(continued from previous page)

```
| 1 | 2 | 3 |
+---+---+---+
| 3 | 4 | 5 |
+---+---+---+
```

Yet, it is possible assign a absolute url to an online excel file to an instance of `pyexcel.Sheet`.

custom attributes

You can pass on source specific parameters to getter and setter functions.

```
>>> content = "1-2-3\n3-4-5"
>>> sheet = pyexcel.Sheet()
>>> sheet.set_csv(content, delimiter="-")
>>> sheet.csv
'1,2,3\r\n3,4,5\r\n'
>>> sheet.get_csv(delimiter="|")
'1|2|3\r\n3|4|5\r\n'
```

Data manipulation

The data in a sheet is represented by `Sheet` which maintains the data as a list of lists. You can regard `Sheet` as a two dimensional array with additional iterators. Random access to individual column and row is exposed by `Column` and `Row`

Column manipulation

Suppose have one data file as the following:

```
>>> sheet = pyexcel.get_sheet(file_name="example.xls", name_columns_by_row=0)
>>> sheet
pyexcel sheet:
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+==========+==========+==========+
| 1        | 4        | 7        |
+----------+----------+----------+
| 2        | 5        | 8        |
+----------+----------+----------+
| 3        | 6        | 9        |
+----------+----------+----------+
```

And you want to update `Column 2` with these data: [11, 12, 13]

```
>>> sheet.column["Column 2"] = [11, 12, 13]
>>> sheet.column[1]
[11, 12, 13]
>>> sheet
pyexcel sheet:
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+==========+==========+==========+
| 1        | 11       | 7        |
+----------+----------+----------+
```

(continues on next page)

(continued from previous page)

```
| 2          | 12         | 8          |
+----------+----------+----------+
| 3          | 13         | 9          |
+----------+----------+----------+
```

Remove one column of a data file

If you want to remove `Column 2`, you can just call:

```
>>> del sheet.column["Column 2"]
>>> sheet.column["Column 3"]
[7, 8, 9]
```

The sheet content will become:

```
>>> sheet
pyexcel sheet:
+----------+----------+
| Column 1 | Column 3 |
+==========+==========+
| 1          | 7          |
+----------+----------+
| 2          | 8          |
+----------+----------+
| 3          | 9          |
+----------+----------+
```

Append more columns to a data file

Continue from previous example. Suppose you want add two more columns to the data file

Column 4	Column 5
10	13
11	14
12	15

Here is the example code to append two extra columns:

```
>>> extra_data = [
...     ["Column 4", "Column 5"],
...     [10, 13],
...     [11, 14],
...     [12, 15]
... ]
>>> sheet2 = pyexcel.Sheet(extra_data)
>>> sheet.column += sheet2
>>> sheet.column["Column 4"]
[10, 11, 12]
>>> sheet.column["Column 5"]
[13, 14, 15]
```

Here is what you will get:

```
>>> sheet
pyexcel sheet:
+----------+----------+----------+----------+
| Column 1 | Column 3 | Column 4 | Column 5 |
+==========+==========+==========+==========+
| 1        | 7        | 10       | 13       |
+----------+----------+----------+----------+
| 2        | 8        | 11       | 14       |
+----------+----------+----------+----------+
| 3        | 9        | 12       | 15       |
+----------+----------+----------+----------+
```

Cherry pick some columns to be removed

Suppose you have the following data:

```
>>> data = [
...     ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'],
...     [1,2,3,4,5,6,7,9],
... ]
>>> sheet = pyexcel.Sheet(data, name_columns_by_row=0)
>>> sheet
pyexcel sheet:
+---+---+---+---+---+---+---+---+
| a | b | c | d | e | f | g | h |
+===+===+===+===+===+===+===+===+
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 |
+---+---+---+---+---+---+---+---+
```

And you want to remove columns named as: ‘a’, ‘c, ‘e’, ‘h’. This is how you do it:

```
>>> del sheet.column['a', 'c', 'e', 'h']
>>> sheet
pyexcel sheet:
+---+---+---+---+
| b | d | f | g |
+===+===+===+===+
| 2 | 4 | 6 | 7 |
+---+---+---+---+
```

What if the headers are in a different row

Suppose you have the following data:

```
>>> sheet
pyexcel sheet:
+----------+----------+----------+
| 1        | 2        | 3        |
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+----------+----------+----------+
| 4        | 5        | 6        |
+----------+----------+----------+
```

The way to name your columns is to use index 1:

```
>>> sheet.name_columns_by_row(1)
```

Here is what you get:

```
>>> sheet
pyexcel sheet:
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+==========+==========+==========+
| 1        | 2        | 3        |
+----------+----------+----------+
| 4        | 5        | 6        |
+----------+----------+----------+
```

Row manipulation

Suppose you have the following data:

```
>>> sheet
pyexcel sheet:
+---+---+---+-------+
| a | b | c | Row 1 |
+---+---+---+-------+
| e | f | g | Row 2 |
+---+---+---+-------+
| 1 | 2 | 3 | Row 3 |
+---+---+---+-------+
```

You can name your rows by column index at 3:

```
>>> sheet.name_rows_by_column(3)
>>> sheet
pyexcel sheet:
+-------+---+---+---+
| Row 1 | a | b | c |
+-------+---+---+---+
| Row 2 | e | f | g |
+-------+---+---+---+
| Row 3 | 1 | 2 | 3 |
+-------+---+---+---+
```

Then you can access rows by its name:

```
>>> sheet.row["Row 1"]
['a', 'b', 'c']
```

Formatting

Previous section has assumed the data is in the format that you want. In reality, you have to manipulate the data types a bit to suit your needs. Hence, formatters comes into the scene. use `format()` to apply formatter immediately.

Note: **int**, **float** and **datetime** values are automatically detected in **csv** files since **pyexcel** version 0.2.2

Convert a column of numbers to strings

Suppose you have the following data:

```
>>> import pyexcel
>>> data = [
...     ["userid","name"],
...     [10120,"Adam"],
...     [10121,"Bella"],
...     [10122,"Cedar"]
... ]
>>> sheet = pyexcel.Sheet(data)
>>> sheet.name_columns_by_row(0)
>>> sheet.column["userid"]
[10120, 10121, 10122]
```

As you can see, *userid* column is of *int* type. Next, let's convert the column to string format:

```
>>> sheet.column.format("userid", str)
>>> sheet.column["userid"]
['10120', '10121', '10122']
```

Cleanse the cells in a spread sheet

Sometimes, the data in a spreadsheet may have unwanted strings in all or some cells. Let's take an example. Suppose we have a spread sheet that contains all strings but it as random spaces before and after the text values. Some field had weird characters, such as " ":

```
>>> data = [
...     ["        Version", "        Comments", "       Author  "],
...     ["  v0.0.1       ", " Release versions","            Eda"],
...     ["  v0.0.2  ", "Useful updates    ", "   Freud"]
... ]
>>> sheet = pyexcel.Sheet(data)
>>> sheet.content
+-----------------+------------------------------+----------------------+
|         Version |         Comments             |        Author   |
+-----------------+------------------------------+----------------------+
|   v0.0.1        |  Release versions            |             Eda |
+-----------------+------------------------------+----------------------+
|   v0.0.2   | Useful updates     |    Freud        |
+-----------------+------------------------------+----------------------+
```

Now try to create a custom cleanse function:

```
.. code-block:: python
```

```
>>> def cleanse_func(v):
...     v = v.replace(" ", "")
...     v = v.rstrip().strip()
...     return v
...
```

Then let's create a `SheetFormatter` and apply it:

```
.. code-block:: python
```

```
>>> sheet.map(cleanse_func)
```

So in the end, you get this:

```
>>> sheet.content
+---------+------------------+--------+
| Version | Comments         | Author |
+---------+------------------+--------+
| v0.0.1  | Release versions | Eda    |
+---------+------------------+--------+
| v0.0.2  | Useful updates   | Freud  |
+---------+------------------+--------+
```

Data filtering

use `filter()` function to apply a filter immediately. The content is modified.

Suppose you have the following data in any of the supported excel formats:

Column 1	Column 2	Column 3
1	4	7
2	5	8
3	6	9

```
>>> import pyexcel
```

```
>>> sheet = pyexcel.get_sheet(file_name="example_series.xls", name_columns_by_row=0)
>>> sheet.content
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+==========+==========+==========+
| 1        | 2        | 3        |
+----------+----------+----------+
| 4        | 5        | 6        |
+----------+----------+----------+
| 7        | 8        | 9        |
+----------+----------+----------+
```

Filter out some data

You may want to filter odd rows and print them in an array of dictionaries:

```
>>> sheet.filter(row_indices=[0, 2])
>>> sheet.content
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+==========+==========+==========+
| 4        | 5        | 6        |
+----------+----------+----------+
```

Let's try to further filter out even columns:

```
>>> sheet.filter(column_indices=[1])
>>> sheet.content
+----------+----------+
| Column 1 | Column 3 |
+==========+==========+
| 4        | 6        |
+----------+----------+
```

Save the data

Let's save the previous filtered data:

```
>>> sheet.save_as("example_series_filter.xls")
```

When you open *example_series_filter.xls*, you will find these data

Column 1	Column 3
2	8

How to filter out empty rows in my sheet?

Suppose you have the following data in a sheet and you want to remove those rows with blanks:

```
>>> import pyexcel as pe
>>> sheet = pe.Sheet([[1,2,3],['','',''],['','',''],[1,2,3]])
```

You can use `pyexcel.filters.RowValueFilter`, which examines each row, return *True* if the row should be filtered out. So, let's define a filter function:

```
>>> def filter_row(row_index, row):
...     result = [element for element in row if element != '']
...     return len(result)==0
```

And then apply the filter on the sheet:

```
>>> del sheet.row[filter_row]
>>> sheet
pyexcel sheet:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
```

2.4.6 Book

You access each cell via this syntax:

```
book[sheet_index][row, column]
```

or:

```
book["sheet_name"][row, column]
```

Suppose you have the following sheets:

And you can randomly access a cell in a sheet:

```
>>> book = pyexcel.get_book(file_name="example.xls")
>>> print(book["Sheet 1"][0,0])
1
>>> print(book[0][0,0]) # the same cell
1
```

Tip: With pyexcel, you can regard single sheet reader as an two dimensional array and multi-sheet excel book reader as a ordered dictionary of two dimensional arrays.

Write multiple sheet excel file

Suppose you have previous data as a dictionary and you want to save it as multiple sheet excel file:

```
>>> content = {
...     'Sheet 1':
...         [
...             [1.0, 2.0, 3.0],
...             [4.0, 5.0, 6.0],
...             [7.0, 8.0, 9.0]
...         ],
...     'Sheet 2':
...         [
...             ['X', 'Y', 'Z'],
...             [1.0, 2.0, 3.0],
...             [4.0, 5.0, 6.0]
...         ],
...     'Sheet 3':
...         [
...             ['O', 'P', 'Q'],
...             [3.0, 2.0, 1.0],
...             [4.0, 3.0, 2.0]
...         ]
... }
>>> book = pyexcel.get_book(bookdict=content)
>>> book.save_as("output.xls")
```

You shall get a xls file

Read multiple sheet excel file

Let's read the previous file back:

```
>>> book = pyexcel.get_book(file_name="output.xls")
>>> sheets = book.to_dict()
>>> for name in sheets.keys():
...     print(name)
Sheet 1
Sheet 2
Sheet 3
```

Get content

```
>>> book_dict = {
...       'Sheet 2':
...           [
...               ['X', 'Y', 'Z'],
...               [1.0, 2.0, 3.0],
...               [4.0, 5.0, 6.0]
...           ],
...       'Sheet 3':
...           [
...               ['O', 'P', 'Q'],
...               [3.0, 2.0, 1.0],
...               [4.0, 3.0, 2.0]
...           ],
...       'Sheet 1':
...           [
...               [1.0, 2.0, 3.0],
...               [4.0, 5.0, 6.0],
...               [7.0, 8.0, 9.0]
...           ]
...   }
>>> book = pyexcel.get_book(bookdict=book_dict)
>>> book
Sheet 1:
+-----+-----+-----+
| 1.0 | 2.0 | 3.0 |
+-----+-----+-----+
| 4.0 | 5.0 | 6.0 |
+-----+-----+-----+
| 7.0 | 8.0 | 9.0 |
+-----+-----+-----+
Sheet 2:
+-----+-----+-----+
| X   | Y   | Z   |
+-----+-----+-----+
| 1.0 | 2.0 | 3.0 |
+-----+-----+-----+
| 4.0 | 5.0 | 6.0 |
+-----+-----+-----+
Sheet 3:
+-----+-----+-----+
| O   | P   | Q   |
+-----+-----+-----+
| 3.0 | 2.0 | 1.0 |
+-----+-----+-----+
| 4.0 | 3.0 | 2.0 |
+-----+-----+-----+
>>> print(book.rst)
Sheet 1:
=  =  =
1  2  3
4  5  6
7  8  9
=  =  =
Sheet 2:
===  ===  ===
```

(continues on next page)

(continued from previous page)

```
X    Y    Z
1.0  2.0  3.0
4.0  5.0  6.0
===  ===  ===
Sheet 3:
===  ===  ===
O    P    Q
3.0  2.0  1.0
4.0  3.0  2.0
===  ===  ===
```

You can get the direct access to underneath stream object. In some situation, it is desired.

```
>>> stream = book.stream.plain
```

The returned stream object has the content formatted in plain format for further reading.

Set content

Surely, you could set content to an instance of `pyexcel.Book`.

```
>>> other_book = pyexcel.Book()
>>> other_book.bookdict = book_dict
>>> print(other_book.plain)
Sheet 1:
1  2  3
4  5  6
7  8  9
Sheet 2:
X    Y    Z
1.0  2.0  3.0
4.0  5.0  6.0
Sheet 3:
O    P    Q
3.0  2.0  1.0
4.0  3.0  2.0
```

You can set via 'xls' attribute too.

```
>>> another_book = pyexcel.Book()
>>> another_book.xls = other_book.xls
>>> print(another_book.mediawiki)
Sheet 1:
{| class="wikitable" style="text-align: left;"
|+ <!-- caption -->
|-
| align="right"| 1 || align="right"| 2 || align="right"| 3
|-
| align="right"| 4 || align="right"| 5 || align="right"| 6
|-
| align="right"| 7 || align="right"| 8 || align="right"| 9
|}
Sheet 2:
{| class="wikitable" style="text-align: left;"
|+ <!-- caption -->
```

(continues on next page)

(continued from previous page)

```
|-
| X || Y || Z
|-
| 1 || 2 || 3
|-
| 4 || 5 || 6
|}
Sheet 3:
{| class="wikitable" style="text-align: left;"
|+ <!-- caption -->
|-
| O || P || Q
|-
| 3 || 2 || 1
|-
| 4 || 3 || 2
|}
```

Access to individual sheets

You can access individual sheet of a book via attribute:

```
>>> book = pyexcel.get_book(file_name="book.xls")
>>> book.sheet3
sheet3:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+
```

or via array notations:

```
>>> book["sheet 1"] # there is a space in the sheet name
sheet 1:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
```

Merge excel books

Suppose you have two excel books and each had three sheets. You can merge them and get a new book:

You also can merge individual sheets:

```
>>> book1 = pyexcel.get_book(file_name="book1.xls")
>>> book2 = pyexcel.get_book(file_name="book2.xlsx")
```

(continues on next page)

(continued from previous page)

```
>>> merged_book = book1 + book2
>>> merged_book = book1["Sheet 1"] + book2["Sheet 2"]
>>> merged_book = book1["Sheet 1"] + book2
>>> merged_book = book1 + book2["Sheet 2"]
```

Manipulate individual sheets

merge sheets into a single sheet

Suppose you want to merge many csv files row by row into a new sheet.

```
>>> import glob
>>> merged = pyexcel.Sheet()
>>> for file in glob.glob("*.csv"):
...     merged.row += pyexcel.get_sheet(file_name=file)
>>> merged.save_as("merged.csv")
```

How do I read a book, process it and save to a new book

Yes, you can do that. The code looks like this:

```
import pyexcel

book = pyexcel.get_book(file_name="yourfile.xls")
for sheet in book:
    # do you processing with sheet
    # do filtering?
    pass
book.save_as("output.xls")
```

What would happen if I save a multi sheet book into "csv" file

Well, you will get one csv file per each sheet. Suppose you have these code:

```
>>> content = {
...     'Sheet 1':
...         [
...             [1.0, 2.0, 3.0],
...             [4.0, 5.0, 6.0],
...             [7.0, 8.0, 9.0]
...         ],
...     'Sheet 2':
...         [
...             ['X', 'Y', 'Z'],
...             [1.0, 2.0, 3.0],
...             [4.0, 5.0, 6.0]
...         ],
...     'Sheet 3':
...         [
...             ['O', 'P', 'Q'],
...             [3.0, 2.0, 1.0],
```

(continues on next page)

(continued from previous page)

```
...             [4.0, 3.0, 2.0]
...         ]
... }
>>> book = pyexcel.Book(content)
>>> book.save_as("myfile.csv")
```

You will end up with three csv files:

```
>>> import glob
>>> outputfiles = glob.glob("myfile_*.csv")
>>> for file in sorted(outputfiles):
...     print(file)
...
myfile__Sheet 1__0.csv
myfile__Sheet 2__1.csv
myfile__Sheet 3__2.csv
```

and their content is the value of the dictionary at the corresponding key

Alternatively, you could use `save_book_as()` function

```
>>> pyexcel.save_book_as(bookdict=content, dest_file_name="myfile.csv")
```

After I have saved my multiple sheet book in csv format, how do I get them back

First of all, you can read them back individual as csv file using *meth:~pyexcel.get_sheet* method. Secondly, the pyexcel can do the magic to load all of them back into a book. You will just need to provide the common name before the separator "__":

```
>>> book2 = pyexcel.get_book(file_name="myfile.csv")
>>> book2
Sheet 1:
+-----+-----+-----+
| 1.0 | 2.0 | 3.0 |
+-----+-----+-----+
| 4.0 | 5.0 | 6.0 |
+-----+-----+-----+
| 7.0 | 8.0 | 9.0 |
+-----+-----+-----+
Sheet 2:
+-----+-----+-----+
| X   | Y   | Z   |
+-----+-----+-----+
| 1.0 | 2.0 | 3.0 |
+-----+-----+-----+
| 4.0 | 5.0 | 6.0 |
+-----+-----+-----+
Sheet 3:
+-----+-----+-----+
| O   | P   | Q   |
+-----+-----+-----+
| 3.0 | 2.0 | 1.0 |
+-----+-----+-----+
| 4.0 | 3.0 | 2.0 |
+-----+-----+-----+
```

2.4.7 Working with databases

How to import an excel sheet to a database using SQLAlchemy

Note: You can find the complete code of this example in examples folder on github

Before going ahead, let's import the needed components and initialize sql engine and table base:

```
>>> import os
>>> import pyexcel as p
>>> from sqlalchemy import create_engine
>>> from sqlalchemy.ext.declarative import declarative_base
>>> from sqlalchemy import Column , Integer, String, Float, Date
>>> from sqlalchemy.orm import sessionmaker
>>> engine = create_engine("sqlite:///birth.db")
>>> Base = declarative_base()
>>> Session = sessionmaker(bind=engine)
```

Let's suppose we have the following database model:

```
>>> class BirthRegister(Base):
...     __tablename__='birth'
...     id=Column(Integer, primary_key=True)
...     name=Column(String)
...     weight=Column(Float)
...     birth=Column(Date)
```

Let's create the table:

```
>>> Base.metadata.create_all(engine)
```

Now here is a sample excel file to be saved to the table:

Here is the code to import it:

```
>>> session = Session() # obtain a sql session
>>> p.save_as(file_name="birth.xls", name_columns_by_row=0, dest_session=session,␣
↪dest_table=BirthRegister)
```

Done it. It is that simple. Let's verify what has been imported to make sure.

```
>>> sheet = p.get_sheet(session=session, table=BirthRegister)
>>> sheet
birth:
+------------+----+-------+--------+
| birth      | id | name  | weight |
+------------+----+-------+--------+
| 2015-02-03 | 1  | Adam  | 3.4    |
+------------+----+-------+--------+
| 2014-11-12 | 2  | Smith | 4.2    |
+------------+----+-------+--------+
```

2.5 Old tutorial

2.5.1 Work with excel files

Warning: The pyexcel DOES NOT consider Fonts, Styles, Formulas and Charts at all. When you load a stylish excel and update it, you definitely will lose all those styles.

Open a csv file

Read a csv file is simple:

```
>>> import pyexcel as p
>>> sheet = p.get_sheet(file_name="example.csv")
>>> sheet
example.csv:
+---+---+---+
| 1 | 4 | 7 |
+---+---+---+
| 2 | 5 | 8 |
+---+---+---+
| 3 | 6 | 9 |
+---+---+---+
```

The same applies to a tsv file:

```
>>> sheet = p.get_sheet(file_name="example.tsv")
>>> sheet
example.tsv:
+---+---+---+
| 1 | 4 | 7 |
+---+---+---+
| 2 | 5 | 8 |
+---+---+---+
| 3 | 6 | 9 |
+---+---+---+
```

Meanwhile, a tab separated file can be read as csv too. You can specify a delimiter parameter.

```
>>> with open('tab_example.csv', 'w') as f:
...     unused = f.write('I\tam\ttab\tseparated\tcsv\n') # for passing doctest
...     unused = f.write('You\tneed\tdelimiter\tparameter\n') # unused is added
>>> sheet = p.get_sheet(file_name="tab_example.csv", delimiter='\t')
>>> sheet
tab_example.csv:
+-----+------+-----------+-----------+-----+
| I   | am   | tab       | separated | csv |
+-----+------+-----------+-----------+-----+
| You | need | delimiter | parameter |     |
+-----+------+-----------+-----------+-----+
```

Add a new row to an existing file

Suppose you have one data file as the following:

And you want to add a new row:

    12, 11, 10

Here is the code:

```
>>> import pyexcel as pe
>>> sheet = pe.get_sheet(file_name="example.xls")
>>> sheet.row += [12, 11, 10]
>>> sheet.save_as("new_example.xls")
>>> pe.get_sheet(file_name="new_example.xls")
pyexcel_sheet1:
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+----------+----------+----------+
| 1        | 4        | 7        |
+----------+----------+----------+
| 2        | 5        | 8        |
+----------+----------+----------+
| 3        | 6        | 9        |
+----------+----------+----------+
| 12       | 11       | 10       |
+----------+----------+----------+
```

Update an existing row to an existing file

Suppose you want to update the last row of the example file as:

    ['N/A', 'N/A', 'N/A']

Here is the sample code:

```
.. code-block:: python
```

```
>>> import pyexcel as pe
>>> sheet = pe.get_sheet(file_name="example.xls")
>>> sheet.row[3] = ['N/A', 'N/A', 'N/A']
>>> sheet.save_as("new_example1.xls")
>>> pe.get_sheet(file_name="new_example1.xls")
pyexcel_sheet1:
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+----------+----------+----------+
| 1        | 4        | 7        |
+----------+----------+----------+
| 2        | 5        | 8        |
+----------+----------+----------+
| N/A      | N/A      | N/A      |
+----------+----------+----------+
```

Add a new column to an existing file

And you want to add a column instead:

    ["Column 4", 10, 11, 12]

Here is the code:

```
>>> import pyexcel as pe
>>> sheet = pe.get_sheet(file_name="example.xls")
>>> sheet.column += ["Column 4", 10, 11, 12]
>>> sheet.save_as("new_example2.xls")
>>> pe.get_sheet(file_name="new_example2.xls")
pyexcel_sheet1:
+----------+----------+----------+----------+
| Column 1 | Column 2 | Column 3 | Column 4 |
+----------+----------+----------+----------+
| 1        | 4        | 7        | 10       |
+----------+----------+----------+----------+
| 2        | 5        | 8        | 11       |
+----------+----------+----------+----------+
| 3        | 6        | 9        | 12       |
+----------+----------+----------+----------+
```

Update an existing column to an existing file

Again let's update "Column 3" with:

    [100, 200, 300]

Here is the sample code:

```
>>> import pyexcel as pe
>>> sheet = pe.get_sheet(file_name="example.xls")
>>> sheet.column[2] = ["Column 3", 100, 200, 300]
>>> sheet.save_as("new_example3.xls")
>>> pe.get_sheet(file_name="new_example3.xls")
pyexcel_sheet1:
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+----------+----------+----------+
| 1        | 4        | 100      |
+----------+----------+----------+
| 2        | 5        | 200      |
+----------+----------+----------+
| 3        | 6        | 300      |
+----------+----------+----------+
```

Alternatively, you could have done like this:

```
>>> import pyexcel as pe
>>> sheet = pe.get_sheet(file_name="example.xls", name_columns_by_row=0)
>>> sheet.column["Column 3"] = [100, 200, 300]
>>> sheet.save_as("new_example4.xls")
>>> pe.get_sheet(file_name="new_example4.xls")
pyexcel_sheet1:
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+----------+----------+----------+
| 1        | 4        | 100      |
+----------+----------+----------+
| 2        | 5        | 200      |
+----------+----------+----------+
| 3        | 6        | 300      |
+----------+----------+----------+
```

How about the same alternative solution to previous row based example? Well, you'd better to have the following kind of data:

And then you want to update "Row 3" with for example:

```
[100, 200, 300]
```

These code would do the job:

```
>>> import pyexcel as pe
>>> sheet = pe.get_sheet(file_name="row_example.xls", name_rows_by_column=0)
>>> sheet.row["Row 3"] = [100, 200, 300]
>>> sheet.save_as("new_example5.xls")
>>> pe.get_sheet(file_name="new_example5.xls")
pyexcel_sheet1:
+-------+-----+-----+-----+
| Row 1 | 1   | 2   | 3   |
+-------+-----+-----+-----+
| Row 2 | 4   | 5   | 6   |
+-------+-----+-----+-----+
| Row 3 | 100 | 200 | 300 |
+-------+-----+-----+-----+
```

2.5.2 Work with excel files in memory

Excel files in memory can be manipulated directly without saving it to physical disk and vice versa. This is useful in excel file handling at file upload or in excel file download. For example:

```
>>> import pyexcel
>>> content = "1,2,3\n3,4,5"
>>> sheet = pyexcel.get_sheet(file_type="csv", file_content=content)
>>> sheet.csv
'1,2,3\r\n3,4,5\r\n'
```

file type as its attributes

Since version 0.3.0, each supported file types became an attribute of the Sheet and Book class. What it means is that:

1. Read the content in memory
2. Set the content in memory

For example, after you have your Sheet and Book instance, you could access its content in a support file type by using its dot notation. The code in previous section could be rewritten as:

```
>>> import pyexcel
>>> content = "1,2,3\n3,4,5"
>>> sheet = pyexcel.Sheet()
>>> sheet.csv = content
>>> sheet.array
[[1, 2, 3], [3, 4, 5]]
```

Read any supported excel and respond its content in json

You can find a real world example in **examples/memoryfile/** directory: pyexcel_server.py. Here is the example snippet

```
1  def upload():
2      if request.method == 'POST' and 'excel' in request.files:
3          # handle file upload
4          filename = request.files['excel'].filename
5          extension = filename.split(".")[-1]
6          # Obtain the file extension and content
7          # pass a tuple instead of a file name
8          content = request.files['excel'].read()
9          if sys.version_info[0] > 2:
10             # in order to support python 3
11             # have to decode bytes to str
12             content = content.decode('utf-8')
13         sheet = pe.get_sheet(file_type=extension, file_content=content)
14         # then use it as usual
15         sheet.name_columns_by_row(0)
16         # respond with a json
17         return jsonify({"result": sheet.dict})
18     return render_template('upload.html')
```

request.files['excel'] in line 4 holds the file object. line 5 finds out the file extension. line 13 obtains a sheet instance. line 15 uses the first row as data header. line 17 sends the json representation of the excel file back to client browser.

Write to memory and respond to download

```
1  data = [
2      [...],
3      ...
4  ]
5
6  @app.route('/download')
7  def download():
8      sheet = pe.Sheet(data)
9      output = make_response(sheet.csv)
10     output.headers["Content-Disposition"] = "attachment; filename=export.csv"
11     output.headers["Content-type"] = "text/csv"
12     return output
```

make_response is a Flask utility to make a memory content as http response.

Note: You can find the corresponding source code at examples/memoryfile

---

Relevant packages

Readily made plugins have been made on top of this example. Here is a list of them:

framework	plugin/middleware/extension
Flask	Flask-Excel
Django	django-excel
Pyramid	pyramid-excel

And you may make your own by using pyexcel-webio

2.5.3 Sheet: Data conversion

How to obtain records from an excel sheet

Suppose you want to process the following excel data :

Here are the example code:

```
>>> import pyexcel as pe
>>> records = pe.get_records(file_name="your_file.xls")
>>> for record in records:
...     print("%s is aged at %d" % (record['Name'], record['Age']))
Adam is aged at 28
Beatrice is aged at 29
Ceri is aged at 30
Dean is aged at 26
```

How to save an python array as an excel file

Suppose you have the following array:

```
>>> data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

And here is the code to save it as an excel file

```
>>> import pyexcel
>>> pyexcel.save_as(array=data, dest_file_name="example.xls")
```

Let's verify it:

```
>>> pyexcel.get_sheet(file_name="example.xls")
pyexcel_sheet1:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+
```

How to save an python array as a csv file with special delimiter

Suppose you have the following array:

```
>>> data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

And here is the code to save it as an excel file

```
>>> import pyexcel
>>> pyexcel.save_as(array=data,
...                 dest_file_name="example.csv",
...                 dest_delimiter=':')
```

Let's verify it:

```
>>> with open("example.csv") as f:
...     for line in f.readlines():
...         print(line.rstrip())
...
1:2:3
4:5:6
7:8:9
```

How to get a dictionary from an excel sheet

Suppose you have a csv, xls, xlsx file as the following:

The following code will give you data series in a dictionary:

```
>>> import pyexcel
>>> from pyexcel._compact import OrderedDict
>>> my_dict = pyexcel.get_dict(file_name="example_series.xls", name_columns_by_row=0)
>>> isinstance(my_dict, OrderedDict)
True
>>> for key, values in my_dict.items():
...     print({str(key): values})
{'Column 1': [1, 4, 7]}
{'Column 2': [2, 5, 8]}
{'Column 3': [3, 6, 9]}
```

Please note that my_dict is an OrderedDict.

How to obtain a dictionary from a multiple sheet book

Suppose you have a multiple sheet book as the following:

Here is the code to obtain those sheets as a single dictionary:

```
>>> import pyexcel
>>> import json
>>> book_dict = pyexcel.get_book_dict(file_name="book.xls")
>>> isinstance(book_dict, OrderedDict)
True
>>> for key, item in book_dict.items():
...     print(json.dumps({key: item}))
{"Sheet 1": [[1, 2, 3], [4, 5, 6], [7, 8, 9]]}
{"Sheet 2": [["X", "Y", "Z"], [1, 2, 3], [4, 5, 6]]}
{"Sheet 3": [["O", "P", "Q"], [3, 2, 1], [4, 3, 2]]}
```

How to save a dictionary of two dimensional array as an excel file

Suppose you want to save the below dictionary to an excel file

```
>>> a_dictionary_of_two_dimensional_arrays = {
...     'Sheet 1':
...         [
...             [1.0, 2.0, 3.0],
...             [4.0, 5.0, 6.0],
...             [7.0, 8.0, 9.0]
```

(continues on next page)

(continued from previous page)

```
...             ],
...         'Sheet 2':
...             [
...                 ['X', 'Y', 'Z'],
...                 [1.0, 2.0, 3.0],
...                 [4.0, 5.0, 6.0]
...             ],
...         'Sheet 3':
...             [
...                 ['O', 'P', 'Q'],
...                 [3.0, 2.0, 1.0],
...                 [4.0, 3.0, 2.0]
...             ]
...     }
```

Here is the code:

```
>>> pyexcel.save_book_as(
...     bookdict=a_dictionary_of_two_dimensional_arrays,
...     dest_file_name="book.xls"
... )
```

If you want to preserve the order of sheets in your dictionary, you have to pass on an ordered dictionary to the function itself. For example:

```
>>> data = OrderedDict()
>>> data.update({"Sheet 2": a_dictionary_of_two_dimensional_arrays['Sheet 2']})
>>> data.update({"Sheet 1": a_dictionary_of_two_dimensional_arrays['Sheet 1']})
>>> data.update({"Sheet 3": a_dictionary_of_two_dimensional_arrays['Sheet 3']})
>>> pyexcel.save_book_as(bookdict=data, dest_file_name="book.xls")
```

Let's verify its order:

```
>>> book_dict = pyexcel.get_book_dict(file_name="book.xls")
>>> for key, item in book_dict.items():
...     print(json.dumps({key: item}))
{"Sheet 2": [["X", "Y", "Z"], [1, 2, 3], [4, 5, 6]]}
{"Sheet 1": [[1, 2, 3], [4, 5, 6], [7, 8, 9]]}
{"Sheet 3": [["O", "P", "Q"], [3, 2, 1], [4, 3, 2]]}
```

Please notice that "Sheet 2" is the first item in the *book_dict*, meaning the order of sheets are preserved.

How to import an excel sheet to a database using SQLAlchemy

Note: You can find the complete code of this example in examples folder on github

Before going ahead, let's import the needed components and initialize sql engine and table base:

```
>>> from sqlalchemy import create_engine
>>> from sqlalchemy.ext.declarative import declarative_base
>>> from sqlalchemy import Column , Integer, String, Float, Date
>>> from sqlalchemy.orm import sessionmaker
>>> engine = create_engine("sqlite:///birth.db")
```

(continues on next page)

(continued from previous page)

```
>>> Base = declarative_base()
>>> Session = sessionmaker(bind=engine)
```

Let's suppose we have the following database model:

```
>>> class BirthRegister(Base):
...     __tablename__='birth'
...     id=Column(Integer, primary_key=True)
...     name=Column(String)
...     weight=Column(Float)
...     birth=Column(Date)
```

Let's create the table:

```
>>> Base.metadata.create_all(engine)
```

Now here is a sample excel file to be saved to the table:

Here is the code to import it:

```
>>> session = Session() # obtain a sql session
>>> pyexcel.save_as(file_name="birth.xls", name_columns_by_row=0, dest_
↪session=session, dest_table=BirthRegister)
```

Done it. It is that simple. Let's verify what has been imported to make sure.

```
>>> sheet = pyexcel.get_sheet(session=session, table=BirthRegister)
>>> sheet
birth:
+------------+----+-------+--------+
| birth      | id | name  | weight |
+------------+----+-------+--------+
| 2015-02-03 | 1  | Adam  | 3.4    |
+------------+----+-------+--------+
| 2014-11-12 | 2  | Smith | 4.2    |
+------------+----+-------+--------+
```

How to open an xls file and save it as csv

Suppose we want to save previous used example 'birth.xls' as a csv file

```
>>> import pyexcel
>>> pyexcel.save_as(file_name="birth.xls", dest_file_name="birth.csv")
```

Again it is really simple. Let's verify what we have gotten:

```
>>> sheet = pyexcel.get_sheet(file_name="birth.csv")
>>> sheet
birth.csv:
+-------+--------+----------+
| name  | weight | birth    |
+-------+--------+----------+
| Adam  | 3.4    | 03/02/15 |
+-------+--------+----------+
| Smith | 4.2    | 12/11/14 |
+-------+--------+----------+
```

Note: Please note that csv(comma separate value) file is pure text file. Formula, charts, images and formatting in xls file will disappear no matter which transcoding tool you use. Hence, pyexcel is a quick alternative for this transcoding job.

How to open an xls file and save it as xlsx

Warning: Formula, charts, images and formatting in xls file will disappear as pyexcel does not support Formula, charts, images and formatting.

Let use previous example and save it as ods instead

```
>>> import pyexcel
>>> pyexcel.save_as(file_name="birth.xls",
...                 dest_file_name="birth.xlsx") # change the file extension
```

Again let's verify what we have gotten:

```
>>> sheet = pyexcel.get_sheet(file_name="birth.xlsx")
>>> sheet
pyexcel_sheet1:
+-------+--------+----------+
| name  | weight | birth    |
+-------+--------+----------+
| Adam  | 3.4    | 03/02/15 |
+-------+--------+----------+
| Smith | 4.2    | 12/11/14 |
+-------+--------+----------+
```

How to open a xls multiple sheet excel book and save it as csv

Well, you write similar codes as before but you will need to use `save_book_as()` function.

2.5.4 Dot notation for data source

Since version 0.3.0, the data source becomes an attribute of the pyexcel native classes. All support data format is a dot notation away.

For sheet

Get content

```
>>> import pyexcel
>>> content = "1,2,3\n3,4,5"
>>> sheet = pyexcel.get_sheet(file_type="csv", file_content=content)
>>> sheet.tsv
'1\t2\t3\r\n3\t4\t5\r\n'
>>> print(sheet.simple)
```

(continues on next page)

(continued from previous page)

```
csv:
-  -  -
1  2  3
3  4  5
-  -  -
```

What's more, you could as well set value to an attribute, for example:

```
>>> import pyexcel
>>> content = "1,2,3\n3,4,5"
>>> sheet = pyexcel.Sheet()
>>> sheet.csv = content
>>> sheet.array
[[1, 2, 3], [3, 4, 5]]
```

You can get the direct access to underneath stream object. In some situation, it is desired.

```
>>> stream = sheet.stream.tsv
```

The returned stream object has tsv formatted content for reading.

Set content

What you could further do is to set a memory stream of any supported file format to a sheet. For example:

```
>>> another_sheet = pyexcel.Sheet()
>>> another_sheet.xls = sheet.xls
>>> another_sheet.content
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 3 | 4 | 5 |
+---+---+---+
```

Yet, it is possible assign a absolute url to an online excel file to an instance of `pyexcel.Sheet`.

```
>>> another_sheet.url = "https://github.com/pyexcel/pyexcel/raw/master/examples/
↪basics/multiple-sheets-example.xls"
>>> another_sheet.content
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+
```

For book

The same dot notation is available to `pyexcel.Book` as well.

Get content

```
>>> book_dict = {
...       'Sheet 2':
...           [
...               ['X', 'Y', 'Z'],
...               [1.0, 2.0, 3.0],
...               [4.0, 5.0, 6.0]
...           ],
...       'Sheet 3':
...           [
...               ['O', 'P', 'Q'],
...               [3.0, 2.0, 1.0],
...               [4.0, 3.0, 2.0]
...           ],
...       'Sheet 1':
...           [
...               [1.0, 2.0, 3.0],
...               [4.0, 5.0, 6.0],
...               [7.0, 8.0, 9.0]
...           ]
...   }
>>> book = pyexcel.get_book(bookdict=book_dict)
>>> book
Sheet 1:
+-----+-----+-----+
| 1.0 | 2.0 | 3.0 |
+-----+-----+-----+
| 4.0 | 5.0 | 6.0 |
+-----+-----+-----+
| 7.0 | 8.0 | 9.0 |
+-----+-----+-----+
Sheet 2:
+-----+-----+-----+
| X   | Y   | Z   |
+-----+-----+-----+
| 1.0 | 2.0 | 3.0 |
+-----+-----+-----+
| 4.0 | 5.0 | 6.0 |
+-----+-----+-----+
Sheet 3:
+-----+-----+-----+
| O   | P   | Q   |
+-----+-----+-----+
| 3.0 | 2.0 | 1.0 |
+-----+-----+-----+
| 4.0 | 3.0 | 2.0 |
+-----+-----+-----+
>>> print(book.rst)
Sheet 1:
=   =   =
1   2   3
4   5   6
7   8   9
=   =   =
Sheet 2:
===  ===  ===
```

(continues on next page)

(continued from previous page)

```
X    Y    Z
1.0  2.0  3.0
4.0  5.0  6.0
===  ===  ===
Sheet 3:
===  ===  ===
O    P    Q
3.0  2.0  1.0
4.0  3.0  2.0
===  ===  ===
```

You can get the direct access to underneath stream object. In some situation, it is desired.

```
>>> stream = sheet.stream.plain
```

The returned stream object has the content formatted in plain format for further reading.

Set content

Surely, you could set content to an instance of `pyexcel.Book`.

```
>>> other_book = pyexcel.Book()
>>> other_book.bookdict = book_dict
>>> print(other_book.plain)
Sheet 1:
1  2  3
4  5  6
7  8  9
Sheet 2:
X    Y    Z
1.0  2.0  3.0
4.0  5.0  6.0
Sheet 3:
O    P    Q
3.0  2.0  1.0
4.0  3.0  2.0
```

You can set via 'xls' attribute too.

```
>>> another_book = pyexcel.Book()
>>> another_book.xls = other_book.xls
>>> print(another_book.mediawiki)
Sheet 1:
{| class="wikitable" style="text-align: left;"
|+ <!-- caption -->
|-
| align="right"| 1 || align="right"| 2 || align="right"| 3
|-
| align="right"| 4 || align="right"| 5 || align="right"| 6
|-
| align="right"| 7 || align="right"| 8 || align="right"| 9
|}
Sheet 2:
{| class="wikitable" style="text-align: left;"
|+ <!-- caption -->
```

(continues on next page)

(continued from previous page)

```
|-
| X || Y || Z
|-
| 1 || 2 || 3
|-
| 4 || 5 || 6
|}
Sheet 3:
{| class="wikitable" style="text-align: left;"
|+ <!-- caption -->
|-
| O || P || Q
|-
| 3 || 2 || 1
|-
| 4 || 3 || 2
|}
```

How about setting content via a url?

```
>>> another_book.url = "https://github.com/pyexcel/pyexcel/raw/master/examples/basics/
↪multiple-sheets-example.xls"
>>> another_book
Sheet 1:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+
Sheet 2:
+---+---+---+
| X | Y | Z |
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
Sheet 3:
+---+---+---+
| O | P | Q |
+---+---+---+
| 3 | 2 | 1 |
+---+---+---+
| 4 | 3 | 2 |
+---+---+---+
```

Getters and Setters

You can pass on source specific parameters to getter and setter functions.

```
>>> content = "1-2-3\n3-4-5"
>>> sheet = pyexcel.Sheet()
>>> sheet.set_csv(content, delimiter="-")
```

(continues on next page)

(continued from previous page)

```
>>> sheet.csv
'1,2,3\r\n3,4,5\r\n'
>>> sheet.get_csv(delimiter="|")
'1|2|3\r\n3|4|5\r\n'
```

2.5.5 Read partial data

Here the code to get an array of data from your excel file.

The following code will give you the data in json:

```
>>> import pyexcel
>>> # "example.csv","example.xlsx","example.xlsm"
>>> my_array = pyexcel.get_array(file_name="example.xls")
>>> my_array
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Pagination

When you are dealing with huge amount of data, e.g. 64GB, obviously you would not like to fill up your memory with those data. Hence pagination feature is developed to read partial data into memory for processing. You can paginate by row, by column and by both.

Please note that *pyexcel-xls*, *pyexcel-xlsx*, *pyexcel-ods* and *pyexcel-ods3* will read all data into memory. And pagination code here only limits the data returned to your program. With that said, csv readers, *pyexcel-xlsxr* and *pyexcel-odsr* DOES read partial data into memory.

Let's assume the following file is a huge csv file:

```
>>> import datetime
>>> import pyexcel as pe
>>> data = [
...     [1, 21, 31],
...     [2, 22, 32],
...     [3, 23, 33],
...     [4, 24, 34],
...     [5, 25, 35],
...     [6, 26, 36]
... ]
>>> pe.save_as(array=data, dest_file_name="your_file.csv")
```

And let's pretend to read partial data:

```
>>> pe.get_sheet(file_name="your_file.csv", start_row=2, row_limit=3)
your_file.csv:
+---+----+----+
| 3 | 23 | 33 |
+---+----+----+
| 4 | 24 | 34 |
+---+----+----+
| 5 | 25 | 35 |
+---+----+----+
```

And you could as well do the same for columns:

```
>>> pe.get_sheet(file_name="your_file.csv", start_column=1, column_limit=2)
your_file.csv:
+----+----+
| 21 | 31 |
+----+----+
| 22 | 32 |
+----+----+
| 23 | 33 |
+----+----+
| 24 | 34 |
+----+----+
| 25 | 35 |
+----+----+
| 26 | 36 |
+----+----+
```

Obvious, you could do both at the same time:

```
>>> pe.get_sheet(file_name="your_file.csv",
...     start_row=2, row_limit=3,
...     start_column=1, column_limit=2)
your_file.csv:
+----+----+
| 23 | 33 |
+----+----+
| 24 | 34 |
+----+----+
| 25 | 35 |
+----+----+
```

The pagination support is available across all pyexcel plugins.

Note: No column pagination support for query sets as data source.

Formatting while transcoding a big data file

If you are transcoding a big data set, conventional formatting method would not help unless a on-demand free RAM is available. However, there is a way to minimize the memory footprint of pyexcel while the formatting is performed.

Let's continue from previous example. Suppose we want to transcode "your_file.csv" to "your_file.xls" but increase each element by 1.

What we can do is to define a row renderer function as the following:

```
>>> def increment_by_one(row):
...     for element in row:
...         yield element + 1
```

Then pass it onto save_as function using row_renderer:

```
>>> pe.isave_as(file_name="your_file.csv",
...             row_renderer=increment_by_one,
...             dest_file_name="your_file.xlsx")
```

Note: If the data content is from a generator, isave_as has to be used.

We can verify if it was done correctly:

```
>>> pe.get_sheet(file_name="your_file.xlsx")
your_file.csv:
+---+----+----+
| 2 | 22 | 32 |
+---+----+----+
| 3 | 23 | 33 |
+---+----+----+
| 4 | 24 | 34 |
+---+----+----+
| 5 | 25 | 35 |
+---+----+----+
| 6 | 26 | 36 |
+---+----+----+
| 7 | 27 | 37 |
+---+----+----+
```

2.5.6 Sheet: Data Access

Iterate a csv file

Here is the way to read the csv file and iterate through each row:

```
>>> sheet = pyexcel.get_sheet(file_name='tutorial.csv')
>>> for row in sheet:
...     print("%s: %s" % (row[0], row[1]))
Name: Age
Chu Chu: 10
Mo mo: 11
```

Often people wanted to use csv.Dict reader to read it because it has a header. Here is how you do it with pyexcel:

```
1 >>> sheet = pyexcel.get_sheet(file_name='tutorial.csv')
2 >>> sheet.name_columns_by_row(0)
3 >>> for row in sheet:
4 ...     print("%s: %s" % (row[0], row[1]))
5 Chu Chu: 10
6 Mo mo: 11
```

Line 2 remove the header from the actual content. The removed header can be used to access its columns using the name itself, for example:

```
>>> sheet.column['Age']
[10, 11]
```

Random access to individual cell

Top left corner of a sheet is (0, 0), meaning both row index and column index start from 0. To randomly access a cell of `Sheet` instance, two syntax are available:

```
sheet[row, column]
```

This syntax helps you iterate the data by row and by column. If you use excel positions, the syntax below help you get the cell instantly without converting alphabet column index to integer:

```
sheet['A1']
```

Please note that with excel positions, top left corner is 'A1'.

For example: suppose you have the following data sheet,

here is the example code showing how you can randomly access a cell:

```
>>> sheet = pyexcel.get_sheet(file_name="example.xls")
>>> sheet.content
+---------+---+---+---+
| Example | X | Y | Z |
+---------+---+---+---+
| a       | 1 | 2 | 3 |
+---------+---+---+---+
| b       | 4 | 5 | 6 |
+---------+---+---+---+
| c       | 7 | 8 | 9 |
+---------+---+---+---+
>>> print(sheet[3, 2])
8
>>> print(sheet["D3"])
6
>>> sheet[2, 3] = 10
>>> print(sheet[2, 3])
10
```

Note: In order to set a value to a cell, please use sheet[row_index, column_index] = new_value

Random access to rows and columns

Continue with previous excel file, you can access row and column separately:

```
>>> sheet.row[1]
['a', 1, 2, 3]
>>> sheet.column[2]
['Y', 2, 5, 8]
```

Use custom names instead of index

Alternatively, it is possible to use the first row to refer to each columns:

```
>>> sheet.name_columns_by_row(0)
>>> print(sheet[1, "Y"])
5
>>> sheet[1, "Y"] = 100
>>> print(sheet[1, "Y"])
100
```

You have noticed the row index has been changed. It is because first row is taken as the column names, hence all rows after the first row are shifted. Now accessing the columns are changed too:

```
>>> sheet.column['Y']
[2, 100, 8]
```

Hence access the same cell, this statement also works:

```
>>> sheet.column['Y'][1]
100
```

Further more, it is possible to use first column to refer to each rows:

```
>>> sheet.name_rows_by_column(0)
```

To access the same cell, we can use this line:

```
>>> sheet.row["b"][1]
100
```

For the same reason, the row index has been reduced by 1. Since we have named columns and rows, it is possible to access the same cell like this:

```
>>> print(sheet["b", "Y"])
100
>>> sheet["b", "Y"] = 200
>>> print(sheet["b", "Y"])
200
```

Note: When you have named your rows and columns, in order to set a value to a cell, please use sheet[row_name, column_name] = new_value

For multiple sheet file, you can regard it as three dimensional array if you use `Book`. So, you access each cell via this syntax:

```
book[sheet_index][row, column]
```

or:

```
book["sheet_name"][row, column]
```

Suppose you have the following sheets:

And you can randomly access a cell in a sheet:

```
>>> book = pyexcel.get_book(file_name="example.xls")
>>> print(book["Sheet 1"][0,0])
1
>>> print(book[0][0,0]) # the same cell
1
```

Tip: With pyexcel, you can regard single sheet reader as an two dimensional array and multi-sheet excel book reader as a ordered dictionary of two dimensional arrays.

Reading a single sheet excel file

Suppose you have a csv, xls, xlsx file as the following:

The following code will give you the data in json:

```
>>> import json
>>> # "example.csv","example.xlsx","example.xlsm"
>>> sheet = pyexcel.get_sheet(file_name="example.xls")
>>> print(json.dumps(sheet.to_array()))
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Read the sheet as a dictionary

Suppose you have a csv, xls, xlsx file as the following:

The following code will give you data series in a dictionary:

```
>>> # "example.xls","example.xlsx","example.xlsm"
>>> sheet = pyexcel.get_sheet(file_name="example_series.xls", name_columns_by_row=0)
```

```
>>> sheet.to_dict()
OrderedDict([('Column 1', [1, 4, 7]), ('Column 2', [2, 5, 8]), ('Column 3', [3, 6,␣
↪9])])
```

Can I get an array of dictionaries per each row?

Suppose you have the following data:

The following code will produce what you want:

```
>>> # "example.csv","example.xlsx","example.xlsm"
>>> sheet = pyexcel.get_sheet(file_name="example.xls", name_columns_by_row=0)
>>> records = sheet.to_records()
>>> for record in records:
...     keys = sorted(record.keys())
...     print("{")
...     for key in keys:
...         print("'%s':%d" % (key, record[key]))
...     print("}")
{
'X':1
'Y':2
'Z':3
}
{
'X':4
'Y':5
'Z':6
}
{
'X':7
'Y':8
'Z':9
}
```

Writing a single sheet excel file

Suppose you have an array as the following:

1	2	3
4	5	6
7	8	9

The following code will write it as an excel file of your choice:

```
.. testcode::
```

```
>>> array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> # "output.xls" "output.xlsx" "output.ods" "output.xlsm"
>>> sheet = pyexcel.Sheet(array)
>>> sheet.save_as("output.csv")
```

Suppose you have a dictionary as the following:

The following code will write it as an excel file of your choice:

```
>>> example_dict = {"Column 1": [1, 2, 3], "Column 2": [4, 5, 6], "Column 3": [7, 8,␣
↪9]}
>>> # "output.xls" "output.xlsx" "output.ods" "output.xlsm"
>>> sheet = pyexcel.get_sheet(adict=example_dict)
>>> sheet.save_as("output.csv")
```

Write multiple sheet excel file

Suppose you have previous data as a dictionary and you want to save it as multiple sheet excel file:

```
>>> content = {
...     'Sheet 1':
...         [
...             [1.0, 2.0, 3.0],
...             [4.0, 5.0, 6.0],
...             [7.0, 8.0, 9.0]
...         ],
...     'Sheet 2':
...         [
...             ['X', 'Y', 'Z'],
...             [1.0, 2.0, 3.0],
...             [4.0, 5.0, 6.0]
...         ],
...     'Sheet 3':
...         [
...             ['O', 'P', 'Q'],
...             [3.0, 2.0, 1.0],
...             [4.0, 3.0, 2.0]
...         ]
... }
>>> book = pyexcel.get_book(bookdict=content)
>>> book.save_as("output.xls")
```

You shall get a xls file

Read multiple sheet excel file

Let's read the previous file back:

```
>>> book = pyexcel.get_book(file_name="output.xls")
>>> sheets = book.to_dict()
>>> for name in sheets.keys():
...     print(name)
Sheet 1
Sheet 2
Sheet 3
```

Work with data series in a single sheet

Suppose you have the following data in any of the supported excel formats again:

```
>>> sheet = pyexcel.get_sheet(file_name="example_series.xls", name_columns_by_row=0)
```

Play with data

You can get headers:

```
>>> print(list(sheet.colnames))
['Column 1', 'Column 2', 'Column 3']
```

You can use a utility function to get all in a dictionary:

```
>>> sheet.to_dict()
OrderedDict([('Column 1', [1, 4, 7]), ('Column 2', [2, 5, 8]), ('Column 3', [3, 6,␣
↪9])])
```

Maybe you want to get only the data without the column headers. You can call `rows()` instead:

```
>>> list(sheet.rows())
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

You can get data from the bottom to the top one by calling `rrows()` instead:

```
>>> list(sheet.rrows())
[[7, 8, 9], [4, 5, 6], [1, 2, 3]]
```

You might want the data arranged vertically. You can call `columns()` instead:

```
>>> list(sheet.columns())
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

You can get columns in reverse sequence as well by calling `rcolumns()` instead:

```
>>> list(sheet.rcolumns())
[[3, 6, 9], [2, 5, 8], [1, 4, 7]]
```

Do you want to flatten the data? You can get the content in one dimensional array. If you are interested in playing with one dimensional enumeration, you can check out these functions `enumerate()`, `reverse()`, `vertical()`, and `rvertical()`:

```
>>> list(sheet.enumerate())
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(sheet.reverse())
[9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> list(sheet.vertical())
[1, 4, 7, 2, 5, 8, 3, 6, 9]
>>> list(sheet.rvertical())
[9, 6, 3, 8, 5, 2, 7, 4, 1]
```

2.5.7 Sheet: Data manipulation

The data in a sheet is represented by `Sheet` which maintains the data as a list of lists. You can regard `Sheet` as a two dimensional array with additional iterators. Random access to individual column and row is exposed by `Column` and `Row`

Column manipulation

Suppose have one data file as the following:

```
>>> sheet = pyexcel.get_sheet(file_name="example.xls", name_columns_by_row=0)
>>> sheet
pyexcel sheet:
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+==========+==========+==========+
| 1        | 4        | 7        |
+----------+----------+----------+
| 2        | 5        | 8        |
+----------+----------+----------+
| 3        | 6        | 9        |
+----------+----------+----------+
```

And you want to update `Column 2` with these data: [11, 12, 13]

```
>>> sheet.column["Column 2"] = [11, 12, 13]
>>> sheet.column[1]
[11, 12, 13]
>>> sheet
pyexcel sheet:
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+==========+==========+==========+
| 1        | 11       | 7        |
+----------+----------+----------+
| 2        | 12       | 8        |
+----------+----------+----------+
| 3        | 13       | 9        |
+----------+----------+----------+
```

Remove one column of a data file

If you want to remove `Column 2`, you can just call:

```
>>> del sheet.column["Column 2"]
>>> sheet.column["Column 3"]
[7, 8, 9]
```

The sheet content will become:

```
>>> sheet
pyexcel sheet:
+----------+----------+
| Column 1 | Column 3 |
+==========+==========+
| 1        | 7        |
+----------+----------+
| 2        | 8        |
+----------+----------+
| 3        | 9        |
+----------+----------+
```

Append more columns to a data file

Continue from previous example. Suppose you want add two more columns to the data file

Column 4	Column 5
10	13
11	14
12	15

Here is the example code to append two extra columns:

```
>>> extra_data = [
...     ["Column 4", "Column 5"],
...     [10, 13],
...     [11, 14],
...     [12, 15]
... ]
>>> sheet2 = pyexcel.Sheet(extra_data)
>>> sheet.column += sheet2
>>> sheet.column["Column 4"]
[10, 11, 12]
>>> sheet.column["Column 5"]
[13, 14, 15]
```

Here is what you will get:

```
>>> sheet
pyexcel sheet:
+----------+----------+----------+----------+
| Column 1 | Column 3 | Column 4 | Column 5 |
+==========+==========+==========+==========+
| 1        | 7        | 10       | 13       |
+----------+----------+----------+----------+
| 2        | 8        | 11       | 14       |
+----------+----------+----------+----------+
| 3        | 9        | 12       | 15       |
+----------+----------+----------+----------+
```

Cherry pick some columns to be removed

Suppose you have the following data:

```
>>> data = [
...     ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'],
...     [1,2,3,4,5,6,7,9],
... ]
>>> sheet = pyexcel.Sheet(data, name_columns_by_row=0)
>>> sheet
pyexcel sheet:
+---+---+---+---+---+---+---+---+
| a | b | c | d | e | f | g | h |
+===+===+===+===+===+===+===+===+
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 |
+---+---+---+---+---+---+---+---+
```

And you want to remove columns named as: 'a', 'c, 'e', 'h'. This is how you do it:

```
>>> del sheet.column['a', 'c', 'e', 'h']
>>> sheet
pyexcel sheet:
+---+---+---+---+
| b | d | f | g |
+===+===+===+===+
| 2 | 4 | 6 | 7 |
+---+---+---+---+
```

What if the headers are in a different row

Suppose you have the following data:

```
>>> sheet
pyexcel sheet:
+----------+----------+----------+
| 1        | 2        | 3        |
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+----------+----------+----------+
| 4        | 5        | 6        |
+----------+----------+----------+
```

The way to name your columns is to use index 1:

```
>>> sheet.name_columns_by_row(1)
```

Here is what you get:

```
>>> sheet
pyexcel sheet:
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+==========+==========+==========+
| 1        | 2        | 3        |
+----------+----------+----------+
```

(continues on next page)

(continued from previous page)

```
| 4          | 5          | 6          |
+------------+------------+------------+
```

Row manipulation

Suppose you have the following data:

```
>>> sheet
pyexcel sheet:
+---+---+---+-------+
| a | b | c | Row 1 |
+---+---+---+-------+
| e | f | g | Row 2 |
+---+---+---+-------+
| 1 | 2 | 3 | Row 3 |
+---+---+---+-------+
```

You can name your rows by column index at 3:

```
>>> sheet.name_rows_by_column(3)
>>> sheet
pyexcel sheet:
+-------+---+---+---+
| Row 1 | a | b | c |
+-------+---+---+---+
| Row 2 | e | f | g |
+-------+---+---+---+
| Row 3 | 1 | 2 | 3 |
+-------+---+---+---+
```

Then you can access rows by its name:

```
>>> sheet.row["Row 1"]
['a', 'b', 'c']
```

2.5.8 Sheet: Data filtering

use `filter()` function to apply a filter immediately. The content is modified.

Suppose you have the following data in any of the supported excel formats:

Column 1	Column 2	Column 3
1	4	7
2	5	8
3	6	9

```
>>> import pyexcel
```

```
>>> sheet = pyexcel.get_sheet(file_name="example_series.xls", name_columns_by_row=0)
>>> sheet.content
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
```

(continues on next page)

(continued from previous page)

```
+==========+==========+==========+
| 1        | 2        | 3        |
+----------+----------+----------+
| 4        | 5        | 6        |
+----------+----------+----------+
| 7        | 8        | 9        |
+----------+----------+----------+
```

Filter out some data

You may want to filter odd rows and print them in an array of dictionaries:

```
>>> sheet.filter(row_indices=[0, 2])
>>> sheet.content
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+==========+==========+==========+
| 4        | 5        | 6        |
+----------+----------+----------+
```

Let's try to further filter out even columns:

```
>>> sheet.filter(column_indices=[1])
>>> sheet.content
+----------+----------+
| Column 1 | Column 3 |
+==========+==========+
| 4        | 6        |
+----------+----------+
```

Save the data

Let's save the previous filtered data:

```
>>> sheet.save_as("example_series_filter.xls")
```

When you open *example_series_filter.xls*, you will find these data

Column 1	Column 3
2	8

How to filter out empty rows in my sheet?

Suppose you have the following data in a sheet and you want to remove those rows with blanks:

```
>>> import pyexcel as pe
>>> sheet = pe.Sheet([[1,2,3],['','',''],['','',''],[1,2,3]])
```

You can use `pyexcel.filters.RowValueFilter`, which examines each row, return *True* if the row should be filtered out. So, let's define a filter function:

```
>>> def filter_row(row_index, row):
...     result = [element for element in row if element != '']
...     return len(result)==0
```

And then apply the filter on the sheet:

```
>>> del sheet.row[filter_row]
>>> sheet
pyexcel sheet:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
```

2.5.9 Sheet: Formatting

Previous section has assumed the data is in the format that you want. In reality, you have to manipulate the data types a bit to suit your needs. Hence, formatters comes into the scene. use `format()` to apply formatter immediately.

Note: **int**, **float** and **datetime** values are automatically detected in **csv** files since **pyexcel** version 0.2.2

Convert a column of numbers to strings

Suppose you have the following data:

```
>>> import pyexcel
>>> data = [
...     ["userid","name"],
...     [10120,"Adam"],
...     [10121,"Bella"],
...     [10122,"Cedar"]
... ]
>>> sheet = pyexcel.Sheet(data)
>>> sheet.name_columns_by_row(0)
>>> sheet.column["userid"]
[10120, 10121, 10122]
```

As you can see, *userid* column is of *int* type. Next, let's convert the column to string format:

```
>>> sheet.column.format("userid", str)
>>> sheet.column["userid"]
['10120', '10121', '10122']
```

Cleanse the cells in a spread sheet

Sometimes, the data in a spreadsheet may have unwanted strings in all or some cells. Let's take an example. Suppose we have a spread sheet that contains all strings but it as random spaces before and after the text values. Some field had weird characters, such as " ":

```
>>> data = [
...     ["        Version", "        Comments", "       Author  "],
...     ["  v0.0.1       ", " Release versions","            Eda"],
...     ["  v0.0.2  ", "Useful updates    ", "   Freud"]
... ]
>>> sheet = pyexcel.Sheet(data)
>>> sheet.content
+-----------------+------------------------------+----------------------+
|         Version |         Comments             |        Author   |
+-----------------+------------------------------+----------------------+
|   v0.0.1        |  Release versions            |             Eda |
+-----------------+------------------------------+----------------------+
|   v0.0.2   | Useful updates     |    Freud        |
+-----------------+------------------------------+----------------------+
```

Now try to create a custom cleanse function:

```
.. code-block:: python
```

```
>>> def cleanse_func(v):
...     v = v.replace(" ", "")
...     v = v.rstrip().strip()
...     return v
...
```

Then let's create a `SheetFormatter` and apply it:

```
.. code-block:: python
```

```
>>> sheet.map(cleanse_func)
```

So in the end, you get this:

```
>>> sheet.content
+---------+-----------------+--------+
| Version | Comments        | Author |
+---------+-----------------+--------+
| v0.0.1  | Release versions | Eda    |
+---------+-----------------+--------+
| v0.0.2  | Useful updates   | Freud  |
+---------+-----------------+--------+
```

2.5.10 Book: Sheet operations

Access to individual sheets

You can access individual sheet of a book via attribute:

```
>>> book = pyexcel.get_book(file_name="book.xls")
>>> book.sheet3
sheet3:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
```

(continues on next page)

(continued from previous page)

```
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+
```

or via array notations:

```
>>> book["sheet 1"] # there is a space in the sheet name
sheet 1:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
```

Merge excel books

Suppose you have two excel books and each had three sheets. You can merge them and get a new book:

You also can merge individual sheets:

```
>>> book1 = pyexcel.get_book(file_name="book1.xls")
>>> book2 = pyexcel.get_book(file_name="book2.xlsx")
>>> merged_book = book1 + book2
>>> merged_book = book1["Sheet 1"] + book2["Sheet 2"]
>>> merged_book = book1["Sheet 1"] + book2
>>> merged_book = book1 + book2["Sheet 2"]
```

Manipulate individual sheets

merge sheets into a single sheet

Suppose you want to merge many csv files row by row into a new sheet.

```
>>> import glob
>>> merged = pyexcel.Sheet()
>>> for file in glob.glob("*.csv"):
...     merged.row += pyexcel.get_sheet(file_name=file)
>>> merged.save_as("merged.csv")
```

How do I read a book, process it and save to a new book

Yes, you can do that. The code looks like this:

```
import pyexcel

book = pyexcel.get_book(file_name="yourfile.xls")
for sheet in book:
    # do you processing with sheet
    # do filtering?
```

(continues on next page)

(continued from previous page)

```
    pass
book.save_as("output.xls")
```

What would happen if I save a multi sheet book into "csv" file

Well, you will get one csv file per each sheet. Suppose you have these code:

```
>>> content = {
...     'Sheet 1':
...         [
...             [1.0, 2.0, 3.0],
...             [4.0, 5.0, 6.0],
...             [7.0, 8.0, 9.0]
...         ],
...     'Sheet 2':
...         [
...             ['X', 'Y', 'Z'],
...             [1.0, 2.0, 3.0],
...             [4.0, 5.0, 6.0]
...         ],
...     'Sheet 3':
...         [
...             ['O', 'P', 'Q'],
...             [3.0, 2.0, 1.0],
...             [4.0, 3.0, 2.0]
...         ]
... }
>>> book = pyexcel.Book(content)
>>> book.save_as("myfile.csv")
```

You will end up with three csv files:

```
>>> import glob
>>> outputfiles = glob.glob("myfile_*.csv")
>>> for file in sorted(outputfiles):
...     print(file)
...
myfile__Sheet 1__0.csv
myfile__Sheet 2__1.csv
myfile__Sheet 3__2.csv
```

and their content is the value of the dictionary at the corresponding key

Alternatively, you could use `save_book_as()` function

```
>>> pyexcel.save_book_as(bookdict=content, dest_file_name="myfile.csv")
```

After I have saved my multiple sheet book in csv format, how do I get them back

First of all, you can read them back individual as csv file using *meth:~pyexcel.get_sheet* method. Secondly, the pyexcel can do the magic to load all of them back into a book. You will just need to provide the common name before the separator "__":

```
>>> book2 = pyexcel.get_book(file_name="myfile.csv")
>>> book2
Sheet 1:
+-----+-----+-----+
| 1.0 | 2.0 | 3.0 |
+-----+-----+-----+
| 4.0 | 5.0 | 6.0 |
+-----+-----+-----+
| 7.0 | 8.0 | 9.0 |
+-----+-----+-----+
Sheet 2:
+-----+-----+-----+
| X   | Y   | Z   |
+-----+-----+-----+
| 1.0 | 2.0 | 3.0 |
+-----+-----+-----+
| 4.0 | 5.0 | 6.0 |
+-----+-----+-----+
Sheet 3:
+-----+-----+-----+
| O   | P   | Q   |
+-----+-----+-----+
| 3.0 | 2.0 | 1.0 |
+-----+-----+-----+
| 4.0 | 3.0 | 2.0 |
+-----+-----+-----+
```

2.6 Cook book

2.6.1 Recipes

Warning: The pyexcel DOES NOT consider Fonts, Styles and Charts at all. In the resulting excel files, fonts, styles and charts will not be transferred.

These recipes give a one-stop utility functions for known use cases. Similar functionality can be achieved using other application interfaces.

Update one column of a data file

Suppose you have one data file as the following:

example.xls

Column 1	Column 2	Column 3
1	4	7
2	5	8
3	6	9

And you want to update `Column 2` with these data: [11, 12, 13]

Here is the code:

```
>>> from pyexcel.cookbook import update_columns
>>> custom_column = {"Column 2":[11, 12, 13]}
>>> update_columns("example.xls", custom_column, "output.xls")
```

Your output.xls will have these data:

Column 1	Column 2	Column 3
1	11	7
2	12	8
3	13	9

Update one row of a data file

Suppose you have the same data file:

example.xls

Row 1	1	2	3
Row 2	4	5	6
Row 3	7	8	9

And you want to update the second row with these data: [7, 4, 1]

Here is the code:

```
>>> from pyexcel.cookbook import update_rows
>>> custom_row = {"Row 1":[11, 12, 13]}
>>> update_rows("example.xls", custom_row, "output.xls")
>>> pyexcel.get_sheet(file_name="output.xls")
pyexcel sheet:
+-------+----+----+----+
| Row 1 | 11 | 12 | 13 |
+-------+----+----+----+
| Row 2 | 4  | 5  | 6  |
+-------+----+----+----+
| Row 3 | 7  | 8  | 9  |
+-------+----+----+----+
```

Merge two files into one

Suppose you want to merge the following two data files:

example.csv

Column 1	Column 2	Column 3
1	4	7
2	5	8
3	6	9

example.xls

Column 4	Column 5
10	12
11	13

The following code will merge the tow into one file, say "output.xls":

```
>>> from pyexcel.cookbook import merge_two_files
>>> merge_two_files("example.csv", "example.xls", "output.xls")
```

The output.xls would have the following data:

Column 1	Column 2	Column 3	Column 4	Column 5
1	4	7	10	12
2	5	8	11	13
3	6	9		

Select candidate columns of two files and form a new one

Suppose you have these two files:

example.ods

Column 1	Column 2	Column 3	Column 4	Column 5
1	4	7	10	13
2	5	8	11	14
3	6	9	12	15

example.xls

Column 6	Column 7	Column 8	Column 9	Column 10
16	17	18	19	20

```
>>> data = [
...       ["Column 1", "Column 2", "Column 3", "Column 4", "Column 5"],
...       [1, 4, 7, 10, 13],
...       [2, 5, 8, 11, 14],
...       [3, 6, 9, 12, 15]
...   ]
>>> s = pyexcel.Sheet(data)
>>> s.save_as("example.csv")
>>> data = [
...       ["Column 6", "Column 7", "Column 8", "Column 9", "Column 10"],
...       [16, 17, 18, 19, 20]
...   ]
>>> s = pyexcel.Sheet(data)
>>> s.save_as("example.xls")
```

And you want to filter out column 2 and 4 from example.ods, filter out column 6 and 7 and merge them:

Column 1	Column 3	Column 5	Column 8	Column 9	Column 10
1	7	13	18	19	20
2	8	14			
3	9	15			

The following code will do the job:

```
>>> from pyexcel.cookbook import merge_two_readers
>>> sheet1 = pyexcel.get_sheet(file_name="example.csv", name_columns_by_row=0)
>>> sheet2 = pyexcel.get_sheet(file_name="example.xls", name_columns_by_row=0)
>>> del sheet1.column[1, 3, 5]
>>> del sheet2.column[0, 1]
>>> merge_two_readers(sheet1, sheet2, "output.xls")
```

Merge two files into a book where each file become a sheet

Suppose you want to merge the following two data files:

example.csv

Column 1	Column 2	Column 3
1	4	7
2	5	8
3	6	9

example.xls

Column 4	Column 5
10	12
11	13

```
>>> data = [
...       ["Column 1", "Column 2", "Column 3"],
...       [1, 2, 3],
...       [4, 5, 6],
...       [7, 8, 9]
...   ]
>>> s = pyexcel.Sheet(data)
>>> s.save_as("example.csv")
>>> data = [
...       ["Column 4", "Column 5"],
...       [10, 12],
...       [11, 13]
...   ]
>>> s = pyexcel.Sheet(data)
>>> s.save_as("example.xls")
```

The following code will merge the tow into one file, say “output.xls”:

```
>>> from pyexcel.cookbook import merge_all_to_a_book
>>> merge_all_to_a_book(["example.csv", "example.xls"], "output.xls")
```

The output.xls would have the following data:

example.csv as sheet name and inside the sheet, you have:

Column 1	Column 2	Column 3
1	4	7
2	5	8
3	6	9

example.ods as sheet name and inside the sheet, you have:

Column 4	Column 5
10	12
11	13

Merge all excel files in directory into a book where each file become a sheet

The following code will merge every excel files into one file, say "output.xls":

```
from pyexcel.cookbook import merge_all_to_a_book
import glob


merge_all_to_a_book(glob.glob("your_csv_directory\*.csv"), "output.xls")
```

You can mix and match with other excel formats: xls, xlsm and ods. For example, if you are sure you have only xls, xlsm, xlsx, ods and csv files in *your_excel_file_directory*, you can do the following:

```
from pyexcel.cookbook import merge_all_to_a_book
import glob


merge_all_to_a_book(glob.glob("your_excel_file_directory\*.*"), "output.xls")
```

Split a book into single sheet files

Suppose you have many sheets in a work book and you would like to separate each into a single sheet excel file. You can easily do this:

```
>>> from pyexcel.cookbook import split_a_book
>>> split_a_book("megabook.xls", "output.xls")
>>> import glob
>>> outputfiles = glob.glob("*_output.xls")
>>> for file in sorted(outputfiles):
...     print(file)
...
Sheet 1_output.xls
Sheet 2_output.xls
Sheet 3_output.xls
```

for the output file, you can specify any of the supported formats

Extract just one sheet from a book

Suppose you just want to extract one sheet from many sheets that exists in a work book and you would like to separate it into a single sheet excel file. You can easily do this:

```
>>> from pyexcel.cookbook import extract_a_sheet_from_a_book
>>> extract_a_sheet_from_a_book("megabook.xls", "Sheet 1", "output.xls")
>>> if os.path.exists("Sheet 1_output.xls"):
...     print("Sheet 1_output.xls exists")
...
Sheet 1_output.xls exists
```

for the output file, you can specify any of the supported formats

2.6.2 Loading from other sources

Get back into pyexcel

list

```
>>> import pyexcel as p
>>> two_dimensional_list = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
>>> sheet = p.get_sheet(array=two_dimensional_list)
>>> sheet
pyexcel_sheet1:
+---+----+----+----+
| 1 | 2  | 3  | 4  |
+---+----+----+----+
| 5 | 6  | 7  | 8  |
+---+----+----+----+
| 9 | 10 | 11 | 12 |
+---+----+----+----+
```

dict

```
>>> a_dictionary_of_key_value_pair = {
...     "IE": 0.2,
...     "Firefox": 0.3
... }
>>> sheet = p.get_sheet(adict=a_dictionary_of_key_value_pair)
>>> sheet
pyexcel_sheet1:
+---------+-----+
| Firefox | IE  |
+---------+-----+
| 0.3     | 0.2 |
+---------+-----+
```

```
>>> a_dictionary_of_one_dimensional_arrays = {
...     "Column 1": [1, 2, 3, 4],
...     "Column 2": [5, 6, 7, 8],
...     "Column 3": [9, 10, 11, 12],
... }
>>> sheet = p.get_sheet(adict=a_dictionary_of_one_dimensional_arrays)
>>> sheet
pyexcel_sheet1:
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+----------+----------+----------+
| 1        | 5        | 9        |
+----------+----------+----------+
| 2        | 6        | 10       |
+----------+----------+----------+
| 3        | 7        | 11       |
+----------+----------+----------+
| 4        | 8        | 12       |
+----------+----------+----------+
```

records

```
>>> a_list_of_dictionaries = [
...     {
...         "Name": 'Adam',
...         "Age": 28
...     },
...     {
...         "Name": 'Beatrice',
...         "Age": 29
...     },
...     {
...         "Name": 'Ceri',
...         "Age": 30
...     },
...     {
...         "Name": 'Dean',
...         "Age": 26
...     }
... ]
>>> sheet = p.get_sheet(records=a_list_of_dictionaries)
>>> sheet
pyexcel_sheet1:
+-----+----------+
| Age | Name     |
+-----+----------+
| 28  | Adam     |
+-----+----------+
| 29  | Beatrice |
+-----+----------+
| 30  | Ceri     |
+-----+----------+
| 26  | Dean     |
+-----+----------+
```

book dict

```
>>> a_dictionary_of_two_dimensional_arrays = {
...       'Sheet 1':
...           [
...               [1.0, 2.0, 3.0],
...               [4.0, 5.0, 6.0],
...               [7.0, 8.0, 9.0]
...           ],
...       'Sheet 2':
...           [
...               ['X', 'Y', 'Z'],
...               [1.0, 2.0, 3.0],
...               [4.0, 5.0, 6.0]
...           ],
...       'Sheet 3':
...           [
...               ['O', 'P', 'Q'],
...               [3.0, 2.0, 1.0],
...               [4.0, 3.0, 2.0]
...           ]
...   }
>>> book = p.get_book(bookdict=a_dictionary_of_two_dimensional_arrays)
>>> book
Sheet 1:
+-----+-----+-----+
| 1.0 | 2.0 | 3.0 |
+-----+-----+-----+
| 4.0 | 5.0 | 6.0 |
+-----+-----+-----+
| 7.0 | 8.0 | 9.0 |
+-----+-----+-----+
Sheet 2:
+-----+-----+-----+
| X   | Y   | Z   |
+-----+-----+-----+
| 1.0 | 2.0 | 3.0 |
+-----+-----+-----+
| 4.0 | 5.0 | 6.0 |
+-----+-----+-----+
Sheet 3:
+-----+-----+-----+
| O   | P   | Q   |
+-----+-----+-----+
| 3.0 | 2.0 | 1.0 |
+-----+-----+-----+
| 4.0 | 3.0 | 2.0 |
+-----+-----+-----+
```

How to load a sheet from a url

Suppose you have excel file somewhere hosted:

```
>>> sheet = pe.get_sheet(url='http://yourdomain.com/test.csv')
>>> sheet
```

(continues on next page)

(continued from previous page)

```
csv:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
```

For sheet

Get content

```
>>> another_sheet = p.Sheet()
>>> another_sheet.url = "https://github.com/pyexcel/pyexcel/raw/master/examples/
↪basics/multiple-sheets-example.xls"
>>> another_sheet.content
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+
```

For book

How about setting content via a url?

```
>>> another_book = p.Book()
>>> another_book.url = "https://github.com/pyexcel/pyexcel/raw/master/examples/basics/
↪multiple-sheets-example.xls"
>>> another_book
Sheet 1:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+
Sheet 2:
+---+---+---+
| X | Y | Z |
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
Sheet 3:
+---+---+---+
| O | P | Q |
+---+---+---+
| 3 | 2 | 1 |
+---+---+---+
```

(continues on next page)

(continued from previous page)

```
| 4 | 3 | 2 |
+---+---+---+
```

2.7 Real world cases

2.7.1 Questions and Answers

1. Python flask writing to a csv file and reading it
2. PyQt: Import .xls file and populate QTableWidget?
3. How do I write data to csv file in columns and rows from a list in python?
4. How to write dictionary values to a csv file using Python
5. Python convert csv to xlsx
6. How to read data from excel and set data type
7. Remove or keep specific columns in csv file
8. How can I put a CSV file in an array?

2.7.2 How to inject csv data to database

Here is real case in the stack-overflow. Due to the author's ignorance, the user would like to have the code in matlab than Python. Hence, I am sharing my pyexcel solution here.

Problem definition

Here is my CSV file:

```
PDB_Id      123442  234335  234336  3549867
a001        6       0       0       8
b001        4       2       0       0
c003        0       0       0       5
```

I want to put this data in a MYSQL table in the form:

```
PROTEIN_ID  PROTEIN_KEY     VALUE_OF_KEY
a001            123442          6
a001            234335          0
a001            234336          0
a001            3549867         8
b001            123442          4
b001            234335          2
b001            234336          0
b001            234336          0
c003            123442          0
c003            234335          0
c003            234336          0
c003            3549867         5
```

I have created table with the following code:

```
sql = """CREATE TABLE ALLPROTEINS (
         Protein_ID CHAR(20),
         PROTEIN_KEY INT ,
         VALUE_OF_KEY INT
         )"""
```

I need the code for insert.

Pyexcel solution

If you could insert an id field to act as the primary key, it can be mapped using sqlalchemy's ORM:

```
$ sqlite3 /tmp/stack2.db
sqlite> CREATE TABLE ALLPROTEINS (
   ...>          ID INT,
   ...>          Protein_ID CHAR(20),
   ...>          PROTEIN_KEY INT,
   ...>          VALUE_OF_KEY INT
   ...>          );
```

Here is the data mapping script vis sqlalchemy:

```
>>> # mapping your database via sqlalchemy
>>> from sqlalchemy import create_engine
>>> from sqlalchemy.ext.declarative import declarative_base
>>> from sqlalchemy import Column, Integer, String
>>> from sqlalchemy.orm import sessionmaker
>>> # checkout http://docs.sqlalchemy.org/en/latest/dialects/index.html
>>> # for a different database server
>>> engine = create_engine("sqlite:////tmp/stack2.db")
>>> Base = declarative_base()
>>> class Proteins(Base):
...     __tablename__ = 'ALLPROTEINS'
...     id = Column(Integer, primary_key=True, autoincrement=True) # <-- appended␣
↪field
...     protein_id = Column(String(20))
...     protein_key = Column(Integer)
...     value_of_key = Column(Integer)
>>> Session = sessionmaker(bind=engine)
>>>
```

Here is the short script to get data inserted into the database:

```
>>> import pyexcel as p
>>> from itertools import product
>>> # data insertion code starts here
>>> sheet = p.get_sheet(file_name="csv-to-mysql-in-matlab-code.csv", delimiter='\t')
>>> sheet.name_columns_by_row(0)
>>> sheet.name_rows_by_column(0)
>>> print(sheet)
csv-to-mysql-in-matlab-code.csv:
+------+--------+--------+--------+---------+
|      | 123442 | 234335 | 234336 | 3549867 |
+======+========+========+========+=========+
| a001 | 6      | 0      | 0      | 8       |
+------+--------+--------+--------+---------+
```

(continues on next page)

(continued from previous page)

```
| b001 | 4       | 2       | 0       | 0        |
+------+---------+---------+---------+----------+
| c003 | 0       | 0       | 0       | 5        |
+------+---------+---------+---------+----------+
>>> results = []
>>> for protein_id, protein_key in product(sheet.rownames, sheet.colnames):
...     results.append([protein_id, protein_key, sheet[str(protein_id), protein_key]])
>>>
>>> sheet2 = p.get_sheet(array=results)
>>> sheet2.colnames = ['protein_id', 'protein_key', 'value_of_key']
>>> print(sheet2)
pyexcel_sheet1:
+------------+-------------+--------------+
| protein_id | protein_key | value_of_key |
+============+=============+==============+
| a001       | 123442      | 6            |
+------------+-------------+--------------+
| a001       | 234335      | 0            |
+------------+-------------+--------------+
| a001       | 234336      | 0            |
+------------+-------------+--------------+
| a001       | 3549867     | 8            |
+------------+-------------+--------------+
| b001       | 123442      | 4            |
+------------+-------------+--------------+
| b001       | 234335      | 2            |
+------------+-------------+--------------+
| b001       | 234336      | 0            |
+------------+-------------+--------------+
| b001       | 3549867     | 0            |
+------------+-------------+--------------+
| c003       | 123442      | 0            |
+------------+-------------+--------------+
| c003       | 234335      | 0            |
+------------+-------------+--------------+
| c003       | 234336      | 0            |
+------------+-------------+--------------+
| c003       | 3549867     | 5            |
+------------+-------------+--------------+
>>> sheet2.save_to_database(session=Session(), table=Proteins)
```

Here is the data inserted:

```
$ sqlite3 /tmp/stack2.db
sqlite> select * from allproteins
   ...> ;
|a001|123442|6
|a001|234335|0
|a001|234336|0
|a001|3549867|8
|b001|123442|4
|b001|234335|2
|b001|234336|0
|b001|234336|0
|c003|123442|0
|c003|234335|0
|c003|234336|0
```

(continues on next page)

(continued from previous page)

```
|c003|3549867|5
```

2.8 API documentation

2.8.1 API Reference

This is intended for users of pyexcel.

Signature functions

Obtaining data from excel file

Saving data to excel file

These flags can be passed on all signature functions:

auto_detect_int

Automatically convert float values to integers if the float number has no decimal values(e.g. 1.00). By default, it does the detection. Setting it to False will turn on this behavior

It has no effect on pyexcel-xlsx because it does that by default.

auto_detect_float

Automatically convert text to float values if possible. This applies only pyexcel-io where csv, tsv, csvz and tsvz formats are supported. By default, it does the detection. Setting it to False will turn on this behavior

auto_detect_datetime

Automatically convert text to python datetime if possible. This applies only pyexcel-io where csv, tsv, csvz and tsvz formats are supported. By default, it does the detection. Setting it to False will turn on this behavior

library

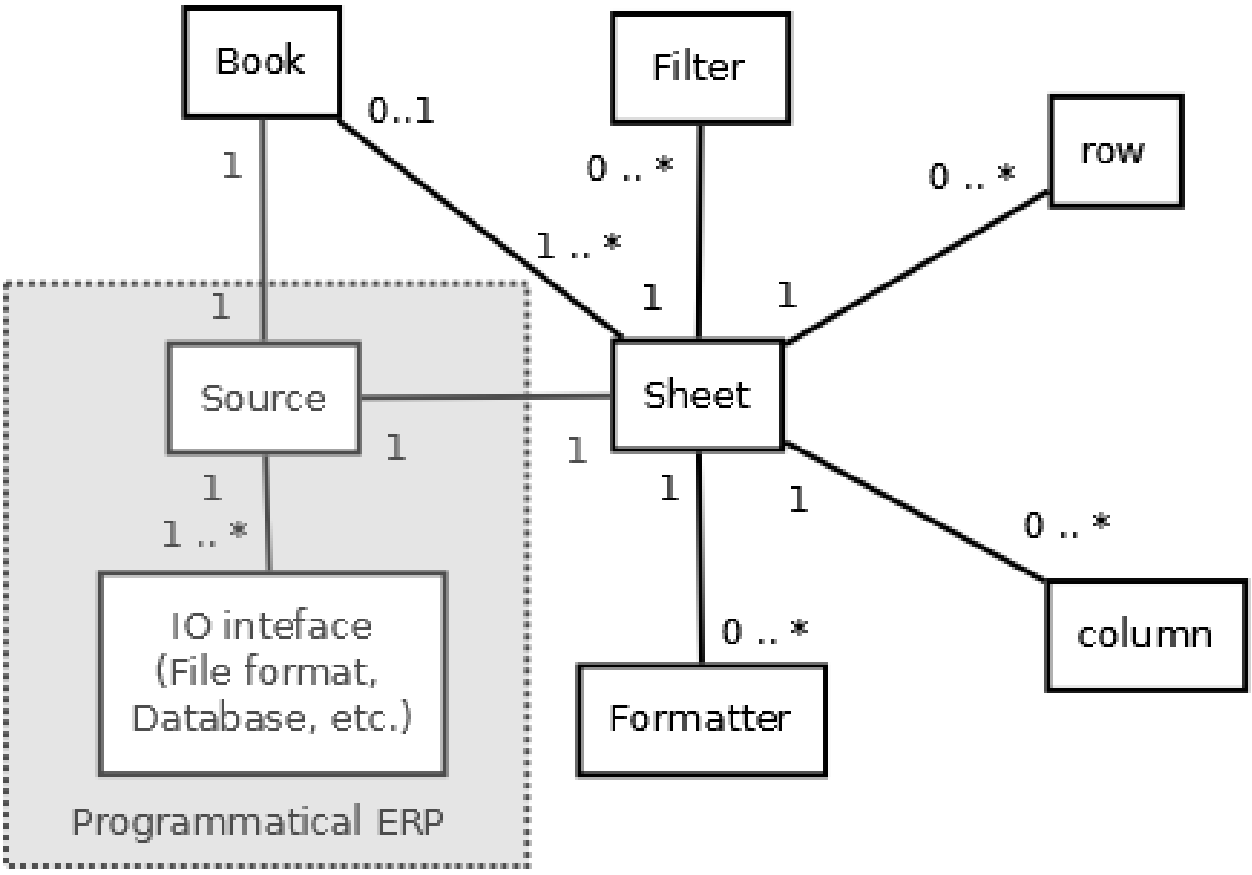
Name a pyexcel plugin to handle a file format. In the situation where multiple plugins were pip installed, it is confusing for pyexcel on which plugin to handle the file format. For example, both pyexcel-xlsx and pyexcel-xls reads xlsx format. Now since version 0.2.2, you can pass on *library="pyexcel-xls"* to handle xlsx in a specific function call.

It is better to uninstall the unwanted pyexcel plugin using pip if two plugins for the same file type are not absolutely necessary.

Cookbook

Book

Here's the entity relationship between Book, Sheet, Row and Column



Constructor

Attribute

Conversions

Save changes

Sheet

Constructor

Attributes

Iteration

Cell access

Row access

Column access

Data series

Any column as row name

Any row as column name

Conversion

Formatting

Filtering

Transformation

Save changes

2.8.2 Internal API reference

This is intended for developers and hackers of pyexcel.

Data sheet representation

In inheritance order from parent to child

Row representation

Column representation

2.9 Developer's guide

2.9.1 Developer's guide

Development steps for code changes

1. git clone https://github.com/pyexcel/pyexcel.git
2. cd pyexcel

Upgrade your setup tools and pip. They are needed for development and testing only:

1. pip install –upgrade setuptools pip

Then install relevant development requirements:

1. pip install -r rnd_requirements.txt # if such a file exists
2. pip install -r requirements.txt
3. pip install -r tests/requirements.txt

Once you have finished your changes, please provide test case(s), relevant documentation and update CHANGELOG.rst.

Note: As to rnd_requirements.txt, usually, it is created when a dependent library is not released. Once the dependecy is installed (will be released), the future version of the dependency in the requirements.txt will be valid.

How to test your contribution

Although *nose* and *doctest* are both used in code testing, it is adviable that unit tests are put in tests. *doctest* is incorporated only to make sure the code examples in documentation remain valid across different development releases.

On Linux/Unix systems, please launch your tests like this:

```
$ make
```

On Windows systems, please issue this command:

```
> test.bat
```

How to update test environment and update documentation

Additional steps are required:

1. pip install moban
2. git clone https://github.com/moremoban/setupmobans.git # generic setup
3. git clone https://github.com/pyexcel/pyexcel-commons.git commons
4. make your changes in *.moban.d* directory, then issue command *moban*

What is pyexcel-commons

Many information that are shared across pyexcel projects, such as: this developer guide, license info, etc. are stored in *pyexcel-commons* project.

What is .moban.d

.moban.d stores the specific meta data for the library.

Acceptance criteria

1. Has Test cases written
2. Has all code lines tested
3. Passes all Travis CI builds
4. Has fair amount of documentation if your change is complex

5. run 'make format' so as to confirm the pyexcel organisation's coding style
6. Please update CHANGELOG.rst
7. Please add yourself to CONTRIBUTORS.rst
8. Agree on NEW BSD License for your contribution

2.9.2 How to log pyexcel

When developing source plugins, it becomes necessary to have log trace available. It helps find out what goes wrong quickly.

The basic step would be to set up logging before pyexcel import statement.

```
import logging
import logging.config
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    level=logging.DEBUG)

import pyexcel
```

And if you would use a complex configuration, you can use the following code.

```
import logging
import logging.config
logging.config.fileConfig('log.conf')

import pyexcel
```

And then save the following content as log.conf in your directory:

```
   [loggers]
   keys=root, sources, renderers

   [handlers]
   keys=consoleHandler

   [formatters]
   keys=custom

   [logger_root]
   level=INFO
   handlers=consoleHandler

   [logger_sources]
   level=DEBUG
   handlers=consoleHandler
   qualname=pyexcel.sources.factory
   propagate=0

   [logger_renderers]
   level=DEBUG
   handlers=consoleHandler
   qualname=pyexcel.renderers.factory
   propagate=0

   [handler_consoleHandler]
```

(continues on next page)

(continued from previous page)

```
   class=StreamHandler
   level=DEBUG
   formatter=custom
   args=(sys.stdout,)

   [formatter_custom]
   format=%(asctime)s - %(name)s - %(levelname)s - %(message)s
   datefmt=


Disable logging
-------------------

In unit testing and django framework, you will find the `lml` logging even you have␣
↪not
explicitly wanted them.


You can suppress them:
```

```
  import logging
  logging.getLogger('lml.plugin').propagate = False

Here is a list of possible modules: `lml.plugin` and `lml.loader`.
```

2.9.3 Packaging with PyInstaller

With pyexcel v0.5.0, the way to package it has been changed because it uses lml for all plugins.

And you need to do the same for pyexcel-io plugins too.

Built-in plugins of pyexcel

In order to package every built-in plugins of pyexcel-io, you need to specify:

```
--hidden-import pyexcel.plugins.renderers.sqlalchemy
--hidden-import pyexcel.plugins.renderers.django
--hidden-import pyexcel.plugins.renderers.excel
--hidden-import pyexcel.plugins.renderers._texttable
--hidden-import pyexcel.plugins.parsers.excel
--hidden-import pyexcel.plugins.parsers.sqlalchemy
--hidden-import pyexcel.plugins.sources.http
--hidden-import pyexcel.plugins.sources.file_input
--hidden-import pyexcel.plugins.sources.memory_input
--hidden-import pyexcel.plugins.sources.file_output
--hidden-import pyexcel.plugins.sources.output_to_memory
--hidden-import pyexcel.plugins.sources.pydata.bookdict
--hidden-import pyexcel.plugins.sources.pydata.dictsource
--hidden-import pyexcel.plugins.sources.pydata.arraysource
--hidden-import pyexcel.plugins.sources.pydata.records
--hidden-import pyexcel.plugins.sources.django
--hidden-import pyexcel.plugins.sources.sqlalchemy
--hidden-import pyexcel.plugins.sources.querysets
```

2.9.4 How to write a plugin for pyexcel

Note: Under writing. Stay tuned.

There are three types of plugins for pyexcel: data parser, data renderer and data source.

Tutorial

Let me walk you through the process of creating pyexcel-pdfr package.

Prerequisites:

1. pip install moban yehua
2. git clone https://github.com/moremoban/setupmobans.git # generic setup
3. git clone https://github.com/pyexcel/pyexcel-commons.git

Let me assume that you have the work directory as:

```
setupmobans pyexcel-commons
```

and *YOUR_WORK_DIRECTORY* points to the base directory for both.

And then please export an environment variable:

```
export YEHUA_FILE=$YOUR_WORK_DIRECTORY/pyexcel-commons/yehua/yehua.yml
```

Now let's get started.

Step 1

Call *yehua* to get the basic scaffolding:

```
$ yehua
Yehua will walk you through creating a pyexcel package.
Press ^C to quit at any time.

What is your project name? pyexcel-pdfr
What is the description? parses tables in pdf file as tabular data
What is project type?
1. pyexcel plugins
2. command line interface
3. python's C externsion
(1,2,3): 1
What is the nick name? pdf
$
```

Step 2

Call *moban* to inflate all project files:

```
$ cd pyexcel-pdfr/
$ ln -s ../pyexcel-commons/ commons
$ ln -s ../setupmobans/ setupmobans
$ moban
Templating README.rst to README.rst
Templating setup.py to setup.py
Templating requirements.txt to requirements.txt
Templating NEW_BSD_LICENSE.jj2 to LICENSE
Templating MANIFEST.in.jj2 to MANIFEST.in
Templating tests/requirements.txt to tests/requirements.txt
Templating test.script.jj2 to test.sh
Templating test.script.jj2 to test.bat
Templating travis.yml.jj2 to .travis.yml
Templating gitignore.jj2 to .gitignore
Templating docs/source/conf.py.jj2 to docs/source/conf.py
```

Step 3 - Coding

Please put your code in pyexcel_pdfr

2.10 Change log

2.10.1 What's breaking in 0.5.9

`pyexcel.Sheet.to_records()` returns a generator instead of a list of dictionaries.

2.10.2 Migrate away from 0.4.3

get_{{file_type}}_stream functions from pyexcel.Sheet and pyexcel.Book were introduced since 0.4.3 but were removed since 0.4.4. Please be advised to use save_to_memory functions, Sheet.io.{{file_type}} or Book.io.{{file_type}}.

2.10.3 Migrate from 0.2.x to 0.3.0+

Filtering and formatting behavior of `pyexcel.Sheet` are simplified. Soft filter and soft formatter are removed. Extra classes such as iterator, formatter, filter are removed.

Most of formatting tasks could be achieved using `format()` and `map()`. and Filtering with `filter()`. Formatting and filtering on row and/or column can be found with `row()` and `column()`

1. Updated filter function

There is no alternative to replace the following code:

```
sheet.filter(pe.OddRowFilter())
```

You will need to remove odd rows by yourself:

```
>>> import pyexcel as pe
>>> data = [
...  ['1'],
...  ['2'],
...  ['3'],
... ]
>>> sheet = pe.Sheet(data)
>>> to_remove = []
>>> for index in sheet.row_range():
...     if index % 2 == 0:
...         to_remove.append(index)
>>> sheet.filter(row_indices=to_remove)
>>> sheet
pyexcel sheet:
+---+
| 2 |
+---+
```

Or, you could do this:

```
>>> data = [
...  ['1'],
...  ['2'],
...  ['3'],
... ]
>>> sheet = pe.Sheet(data)
>>> def odd_filter(row_index, _):
...    return row_index % 2 == 0
>>> del sheet.row[odd_filter]
>>> sheet
pyexcel sheet:
+---+
| 2 |
+---+
```

And the same applies to EvenRowFilter, OddColumnFilter, EvenColumnFilter.

2. Updated format function

2.1 Replacement of sheetformatter

The following formatting code:

```
sheet.apply_formatter(pe.sheets.formatters.SheetFormatter(int))
```

can be replaced by:

```
sheet.format(int)
```

2.2 Replacement of row formatters

The following code:

```
row_formatter = pe.sheets.formatters.RowFormatter([1, 2], str)
sheet.add_formatter(row_formatter)
```

can be replaced by:

```
sheet.row.format([1, 2], str)
```

2.3 Replacement of column formatters

The following code:

```
f = NamedColumnFormatter(["Column 1", "Column 3"], str)
sheet.apply_formatter(f)
```

can be replaced by:

```
sheet.column.format(["Column 1", "Column 3"], str)
```

2.10.4 Migrate from 0.2.1 to 0.2.2+

1. Explicit imports, no longer needed

Please forget about these statements:

```
import pyexcel.ext.xls
import pyexcel.ext.ods
import pyexcel.ext.xlsx
```

They are no longer needed. As long as you have pip-installed them, they will be auto-loaded. However, if you do not want some of the plugins, please use *pip* to uninstall them.

What if you have your code as it is? No harm but a few warnings shown:

```
Deprecated usage since v0.2.2! Explicit import is no longer required. pyexcel.ext.ods␣
↪is auto imported.
```

2. Invalid environment marker: platform_python_implementation=="PyPy"

Yes, it is a surprise. Please upgrade setuptools in your environment:

```
pip install --upgrade setuptools
```

At the time of writing, setuptools (18.0.1) or setuptools-21.0.0-py2.py3-none-any.whl is installed on author's computer and worked.

3. How to keep both pyexcel-xls and pyexcel-xlsx

As in Issue 20, pyexcel-xls was used for xls and pyexcel-xlsx had to be used for xlsx. Both must co-exist due to requirements. The workaround would failed when auto-import are enabled in v0.2.2. Hence, user of pyexcel in this situation shall use 'library' parameter to all signature functions, to instruct pyexcel to use a named library for each function call.

4. pyexcel.get_io is no longer exposed

pyexcel.get_io was passed on from pyexcel-io. However, it is no longer exposed. Please use pyexcel_io.manager.RWManager.get_io if you have to.

You are likely to use pyexcel.get_io when you do `pyexcel.Sheet.save_to_memory()` or `pyexcel.Book.save_to_memory()` where you need to put in a io stream. But actually, with latest code, you could put in a *None*.

2.10.5 Migrate from 0.1.x to 0.2.x

1. "Writer" is gone, Please use save_as.

Here is a piece of legacy code:

```
w = pyexcel.Writer("afile.csv")
data=[['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 1.1, 1]]
w.write_array(table)
w.close()
```

The new code is:

```
>>> data=[['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 1.1, 1]]
>>> pyexcel.save_as(array=data, dest_file_name="afile.csv")
```

Here is another piece of legacy code:

```
content = {
    "X": [1,2,3,4,5],
    "Y": [6,7,8,9,10],
    "Z": [11,12,13,14,15],
}
w = pyexcel.Writer("afile.csv")
w.write_dict(self.content)
w.close()
```

The new code is:

```
>>> content = {
...     "X": [1,2,3,4,5],
...     "Y": [6,7,8,9,10],
...     "Z": [11,12,13,14,15],
... }
>>> pyexcel.save_as(adict=content, dest_file_name="afile.csv")
```

Here is yet another piece of legacy code:

```
data = [
    [1, 2, 3],
    [4, 5, 6]
]
io = StringIO()
w = pyexcel.Writer(("csv",io))
w.write_rows(data)
w.close()
```

The new code is:

```
>>> data = [
...     [1, 2, 3],
...     [4, 5, 6]
... ]
>>> io = pyexcel.save_as(dest_file_type='csv', array=data)
>>> for line in io.readlines():
...     print(line.rstrip())
1,2,3
4,5,6
```

2. "BookWriter" is gone. Please use save_book_as.

Here is a piece of legacy code:

```
import pyexcel
content = {
         "Sheet1": [[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3]],
         "Sheet2": [[4, 4, 4, 4], [5, 5, 5, 5], [6, 6, 6, 6]],
         "Sheet3": [[u'X', u'Y', u'Z'], [1, 4, 7], [2, 5, 8], [3, 6, 9]]
     }
w = pyexcel.BookWriter("afile.csv")
w.write_book_from_dict(content)
w.close()
```

The replacement code is:

```
>>> import pyexcel
>>> content = {
...           "Sheet1": [[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3]],
...           "Sheet2": [[4, 4, 4, 4], [5, 5, 5, 5], [6, 6, 6, 6]],
...           "Sheet3": [[u'X', u'Y', u'Z'], [1, 4, 7], [2, 5, 8], [3, 6, 9]]
...       }
>>> pyexcel.save_book_as(bookdict=content, dest_file_name="afile.csv")
```

2.10.6 Change log

0.5.15 - 07.07.2019

updated

1. #185: fix a bug with http data source. The real fix lies in pyexcel-io v0.5.19. this release just put the version requirement in.

0.5.14 - 12.06.2019

updated

1. #182: support dest_force_file_type on save_as and save_book_as

0.5.13 - 12.03.2019

updated

1. #176: get_sheet {IndexError}list index out of range // XLSX can't be opened

0.5.12 - 25.02.2019

updated

1. #174: include examples in tarbar

0.5.11 - 22.02.2019

updated

1. #169: remove pyexcel-handsontalbe in test
2. add tests, and docs folder in distribution

0.5.10 - 3.12.2018

updated

1. #157: Please use scan_plugins_regex, which lml 0.7 complains about
2. updated dependency on pyexcel-io to 0.5.11

0.5.9.1 - 30.08.2018

updated

1. to require pyexcel-io 0.5.9.1 and use lml at least version 0.0.2

0.5.9 - 30.08.2018

added

1. support __len__. len(book) returns the number of sheets and len(sheet) returns the number of rows
2. #144: memory-efficient way to read sheet names.
3. #148: force_file_type is introduced. When reading a file on a disk, this parameter allows you to choose a reader. i.e. csv reader for a text file. xlsx reader for a xlsx file but with .blob file suffix.
4. finally, pyexcel got import pyexcel.__version__

updated

1. Sheet.to_records() returns a generator now, saving memory
2. #115, Fix set membership test to run faster in python2
3. #140, Direct writes to cells yield weird results

0.5.8 - unreleased

added

1. #125, sort book sheets

updated

1. #126, dest_sheet_name in save_as will set the sheet name in the output
2. #115, Fix set membership test to run faster in python2

0.5.7 - 11.01.2018

added

1. pyexcel-io#46, expose *bulk_save* to developer.

0.5.6 - 23.10.2017

removed

1. #105, remove gease from setup_requires, introduced by 0.5.5.
2. removed testing against python 2.6
3. #103, include LICENSE file in MANIFEST.in, meaning LICENSE file will appear in the released tar ball.

0.5.5 - 20.10.2017

removed

1. #105, remove gease from setup_requires, introduced by 0.5.5.
2. removed testing against python 2.6
3. #103, include LICENSE file in MANIFEST.in, meaning LICENSE file will appear in the released tar ball.

0.5.4 - 27.09.2017

fixed

1. #100, Sheet.to_dict() gets out of range error because there is only one row.

updated

1. Updated the baseline of pyexcel-io to 0.5.1.

0.5.3 - 01-08-2017

added

1. #95, respect the order of records in iget_records, isave_as and save_as.
2. #97, new feature to allow intuitive initialization of pyexcel.Book.

0.5.2 - 26-07-2017

Updated

1. embeded the enabler for pyexcel-htmlr. http source does not support text/html as mime type.

0.5.1 - 12.06.2017

Updated

1. support saving SheetStream and BookStream to database targets. This is needed for pyexcel-webio and its downstream projects.

0.5.0 - 19.06.2017

Added

1. Sheet.top() and Sheet.top_left() for data browsing
2. add html as default rich display in Jupyter notebook when pyexcel-text and pyexcel-chart is installed
3. add svg as default rich display in Jupyter notebook when pyexcel-chart and one of its implementation plugin(pyexcel-pygal, etc.) are is installed
4. new dictionary source supported: a dictionary of key value pair could be read into a sheet.
5. added dynamic external plugin loading. meaning if a pyexcel plugin is installed, it will be loaded implicitly. And this change would remove unnecessary info log for those who do not use pyexcel-text and pyexcel-gal
6. save_book_as before 0.5.0 becomes isave_book_as and save_book_as in 0.5.0 convert BookStream to Book before saving.
7. #83, file closing mechanism is enfored. free_resource is added and it should be called when iget_array, iget_records, isave_as and/or isave_book_as are used.

Updated

1. array is passed to pyexcel.Sheet as reference. it means your array data will be modified.

Removed

1. pyexcel.Writer and pyexcel.BookWriter were removed
2. pyexcel.load_book_from_sql and pyexcel.load_from_sql were removed
3. pyexcel.deprecated.load_from_query_sets, pyexcel.deprecated.load_book_from_django_models and pyexcel.deprecated.load_from_django_model were removed
4. Removed plugin loading code and lml is used instead

0.4.5 - 17.03.2017

Updated

1. #80: remove pyexcel-chart import from v0.4.x

0.4.4 - 06.02.2017

Updated

1. #68: regression save_to_memory() should have returned a stream instance which has been reset to zero if possible. The exception is sys.stdout, which cannot be reset.
2. #74: Not able to handle decimal.Decimal

Removed

1. remove get_{{file_type}}_stream functions from pyexcel.Sheet and pyexcel.Book introduced since 0.4.3.

0.4.3 - 26.01.2017

Added

1. '.stream' attribute are attached to *~pyexcel.Sheet* and *~pyexcel.Book* to get direct access the underneath stream in responding to file type attributes, such as sheet.xls. it helps provide a custom stream to external world, for example, Sheet.stream.csv gives a text stream that contains csv formatted data. Book.stream.xls returns a xls format data in a byte stream.

Updated

1. Better error reporting when an unknown parameters or unsupported file types were given to the signature functions.

0.4.2 - 17.01.2017

Updated

1. Raise exception if the incoming sheet does not have column names. In other words, only sheet with column names could be saved to database. sheet with row names cannot be saved. The alternative is to transpose the sheet, then name_columns_by_row and then save.
2. fix iget_records where a non-uniform content should be given, e.g. [["x", "y"], [1, 2], [3]], some record would become non-uniform, e.g. key 'y' would be missing from the second record.
3. *skip_empty_rows* is applicable when saving a python data structure to another data source. For example, if your array contains a row which is consisted of empty string, such as ['', '', '' ... ''], please specify *skip_empty_rows=False* in order to preserve it. This becomes subtle when you try save a python dictionary where empty rows is not easy to be spotted.
4. #69: better documentation for save_book_as.

0.4.1 - 23.12.2016

Updated

1. #68: regression save_to_memory() should have returned a stream instance.

0.4.0 - 22.12.2016

Added

1. Flask-Excel#19 allow sheet_name parameter
2. pyexcel-xls#11 case-insensitive for file_type. *xls* and *XLS* are treated in the same way

Updated

1. #66: *export_columns* is ignored
2. Update dependency on pyexcel-io v0.3.0

0.3.3 - 07.11.2016

Updated

1. #63: cannot display empty sheet(hence book with empty sheet) as texttable

0.3.2 - 02.11.2016

Updated

1. #62: optional module import error become visible.

0.3.0 - 28.10.2016

Added:

1. file type setters for Sheet and Book, and its documentation
2. *iget_records* returns a generator for a list of records and should have better memory performance, especially dealing with large csv files.
3. *iget_array* returns a generator for a list of two dimensional array and should have better memory performance, especially dealing with large csv files.
4. Enable pagination support, and custom row renderer via pyexcel-io v0.2.3

Updated

1. Take *isave_as* out from *save_as*. Hence two functions are there for save a sheet as
2. #60: encode 'utf-8' if the console is of ascii encoding.
3. #59: custom row renderer
4. #56: set cell value does not work
5. pyexcel.transpose becomes *pyexcel.sheets.transpose*
6. iterator functions of *pyexcel.Sheet* were converted to generator functions
   - *pyexcel.Sheet.enumerate()*
   - *pyexcel.Sheet.reverse()*
   - *pyexcel.Sheet.vertical()*
   - *pyexcel.Sheet.rvertical()*
   - *pyexcel.Sheet.rows()*
   - *pyexcel.Sheet.rrows()*
   - *pyexcel.Sheet.columns()*
   - *pyexcel.Sheet.rcolumns()*
   - *pyexcel.Sheet.named_rows()*
   - *pyexcel.Sheet.named_columns()*
7. *~pyexcel.Sheet.save_to_memory* and *~pyexcel.Book.save_to_memory* return the actual content. No longer they will return a io object hence you cannot call getvalue() on them.

Removed:

1. *content* and *out_file* as function parameters to the signature functions are no longer supported.
2. SourceFactory and RendererFactory are removed
3. The following methods are removed
   - *pyexcel.to_array*
   - *pyexcel.to_dict*
   - *pyexcel.utils.to_one_dimensional_array*

- *pyexcel.dict_to_array*
- *pyexcel.from_records*
- *pyexcel.to_records*

4. *pyexcel.Sheet.filter* has been re-implemented and all filters were removed:
    - *pyexcel.filters.ColumnIndexFilter*
    - *pyexcel.filters.ColumnFilter*
    - *pyexcel.filters.RowFilter*
    - *pyexcel.filters.EvenColumnFilter*
    - *pyexcel.filters.OddColumnFilter*
    - *pyexcel.filters.EvenRowFilter*
    - *pyexcel.filters.OddRowFilter*
    - *pyexcel.filters.RowIndexFilter*
    - *pyexcel.filters.SingleColumnFilter*
    - *pyexcel.filters.RowValueFilter*
    - *pyexcel.filters.NamedRowValueFilter*
    - *pyexcel.filters.ColumnValueFilter*
    - *pyexcel.filters.NamedColumnValueFilter*
    - *pyexcel.filters.SingleRowFilter*
5. the following functions have been removed
    - *add_formatter*
    - *remove_formatter*
    - *clear_formatters*
    - *freeze_formatters*
    - *add_filter*
    - *remove_filter*
    - *clear_filters*
    - *freeze_formatters*
6. *pyexcel.Sheet.filter* has been re-implemented and all filters were removed:
    - pyexcel.formatters.SheetFormatter

0.2.5 - 31.08.2016

Updated:

1. #58: texttable should have been made as compulsory requirement

0.2.4 - 14.07.2016

Updated:

1. For python 2, writing to sys.stdout by pyexcel-cli raise IOError.

0.2.3 - 11.07.2016

Updated:

1. For python 3, do not seek 0 when saving to memory if sys.stdout is passed on. Hence, adding support for sys.stdin and sys.stdout.

0.2.2 - 01.06.2016

Updated:

1. Explicit imports, no longer needed
2. Depends on latest setuptools 18.0.1
3. NotImplementedError will be raised if parameters to core functions are not supported, e.g. get_sheet(cannot_find_me_option="will be thrown out as NotImplementedError")

0.2.1 - 23.04.2016

Added:

1. add pyexcel-text file types as attributes of pyexcel.Sheet and pyexcel.Book, related to #31
2. auto import pyexcel-text if it is pip installed

Updated:

1. code refactoring done for easy addition of sources.
2. bug fix #29, Even if the format is a string it is displayed as a float
3. pyexcel-text is no longer a plugin to pyexcel-io but to pyexcel.sources, see pyexcel-text#22

Removed:

1. pyexcel.presentation is removed. No longer the internal decorate @outsource is used. related to #31

0.2.0 - 17.01.2016

Updated

1. adopt pyexcel-io yield key word to return generator as content
2. pyexcel.save_as and pyexcel.save_book_as get performance improvements

0.1.7 - 03.07.2015

Added

1. Support pyramid-excel which does the database commit on its own.

0.1.6 - 13.06.2015

Added

1. get excel data from a http url

0.0.13 - 07.02.2015

Added

1. Support django
2. texttable as default renderer

0.0.12 - 25.01.2015

Added

1. Added sqlalchemy support

0.0.10 - 15.12.2015

Added

1. added csvz and tsvz format

0.0.4 - 12.10.2014

Updated

1. Support python 3

0.0.1 - 14.09.2014

Features:

1. read and write csv, ods, xls, xlsx and xlsm files(which are referred later as excel files)
2. various iterators for the reader
3. row and column filters for the reader
4. utilities to get array and dictionary out from excel files.

5. cookbok receipes for some common and simple usage of this library.

2.10.7 Note on pypy and lxml

When executing the unit tests on pypy3, it's found that lxml 4.0.0 works well on pypy3. lxml 3.6.4 works well on pypy2.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Bibliography

[f1] quoted from whatis.com. Technical details can be found at MSDN XLS

[f2] xlsx is used by MS-Excel 2007, more information can be found at MSDN XLSX