xlwings - Make Excel Fly!

dev

Zoomer Analytics LLC

2021 03 05

Getting Started

CHAPTER 1

https://training.xlwings.org/p/xlwings

( ) xlwings :)

CHAPTER 2

2.1 Prerequisites

- xlwings requires an **installation of Excel** and therefore only works on **Windows** and **macOS**. Note that macOS currently does not support UDFs.
- xlwings requires at least Python 3.6.

Here are the last versions of xlwings to support:

- Python 3.5: 0.19.5
- Python 2.7: 0.16.6

2.2

xlwings comes pre-installed with

- Anaconda (Windows and macOS)
- WinPython (Windows only) Make sure **not** to take the `dot` version as this only contains Python.

If you are new to Python or have trouble installing xlwings, one of these distributions is highly recommended. Otherwise, you can also install it manually with pip:

```
pip install xlwings
```

conda:

```
conda install xlwings
```

Note that the official `conda` package might be a few releases behind. You can, however, use the `conda-forge` channel (replace `install` with `upgrade` if xlwings is already installed):

```
conda install -c conda-forge xlwings
```

: When you are on macOS and are installing xlwings with `conda` (or use the version that comes with Anaconda), you'll need to run `$ xlwings runpython install` once to enable the `RunPython` calls from VBA. This is done automatically if you install the addin via `$ xlwings addin install`.

2.3

To install the add-in, run the following command:

```
xlwings addin install
```

To call Excel from Python, you don't need an add-in. Also, you can use a single file VBA module (*standalone workbook*) instead of the add-in. For more details, see *Add-in & Settings*.

: The add-in needs to be the same version as the Python package. Make sure to re-install the add-in after upgrading the xlwings package.

2.4

- **Windows**: `pywin32`
- **Mac**: `psutil`, `appscript`

The dependencies are automatically installed via `conda` or `pip`.

2.5 How to activate xlwings PRO

xlwings PRO offers access to *additional functionality*. All PRO features are marked with xlwings *PRO* in the docs.

: To get access to the additional functionality of xlwings PRO, you need a license key and at least xlwings v0.19.0. Everything under the `xlwings.pro` subpackage is distributed under a commercial license. See *xlwings PRO* for more details.

To activate the license key, run the following command:

```
xlwings license update -k LICENSE_KEY
```

Make sure to replace `LICENSE_KEY` with your personal key. This will store the license key under your `xlwings.conf` file (see *User Config: Ribbon/Config File* for where this is on your system). Alternatively, you can also store the license key as an environment variable with the name `XLWINGS_LICENSE_KEY`.

xlwings PRO requires additionally the `cryptography` and `Jinja2` packages which come preinstalled with Anaconda and WinPython. Otherwise, install them via pip or conda.

With pip, you can also run `pip install "xlwings[pro]"` which will take care of the extra dependencies for xlwings PRO.

2.6

- NumPy
- Pandas
- Matplotlib
- Pillow/PIL
- Flask (for REST API)
- cryptography (for xlwings.pro)
- Jinja2 (for xlwings.pro.reports)

These packages are not required but highly recommended as they play very nicely with xlwings. They are all pre-installed with Anaconda. With pip, you can install xlwings with all optional dependencies as follows:

```
pip install "xlwings[all]"
```

2.7 Update

To update to the latest xlwings version, run the following in a command prompt:

```
pip install --upgrade xlwings
```

or:

```
conda update -c conda-forge xlwings
```

Make sure to keep your version of the Excel add-in in sync with your Python package by running the following (make sure to close Excel first):

```
xlwings addin install
```

2.8 Uninstall

To uninstall xlwings completely, first uninstall the add-in, then uninstall the xlwings package using the same method (pip or conda) that you used for installing it:

```
xlwings addin remove
```

Then

```
pip uninstall xlwings
```

or:

```
conda remove xlwings
```

Finally, manually remove the *.xlwings* directory in your home folder if it exists.

CHAPTER 3

xlwings

3.1 1. Interacting with Excel from a Jupyter notebook

If you're just interested in getting a pandas DataFrame in and out of your Jupyter notebook, you can use the `view` and `load` functions, see *Jupyter Notebooks: Interact with Excel*.

3.2 2. Scripting: Automate/interact with Excel from Python

(workbook)

```
>>> import xlwings as xw
>>> wb = xw.Book()  # this will create a new workbook
>>> wb = xw.Book('FileName.xlsx')  # connect to a file that is open or in the␣
↪current working directory
>>> wb = xw.Book(r'C:\path\to\file.xlsx')  # on Windows: use raw strings to␣
↪escape backslashes
```

Excel Excel app `xw.apps.keys()` app (PID)

```
>>> xw.apps[10559].books['FileName.xlsx']
```

(sheet)

```
>>> sht = wb.sheets['Sheet1']
```

(range) /

```
>>> sht.range('A1').value = 'Foo 1'
>>> sht.range('A1').value
'Foo 1'
```

```
>>> sht.range('A1').value = [['Foo 1', 'Foo 2', 'Foo 3'], [10.0, 20.0, 30.0]]
>>> sht.range('A1').expand().value
[['Foo 1', 'Foo 2', 'Foo 3'], [10.0, 20.0, 30.0]]
```

Numpy arrays Pandas DataFrames

```
>>> import pandas as pd
>>> df = pd.DataFrame([[1,2], [3,4]], columns=['a', 'b'])
>>> sht.range('A1').value = df
>>> sht.range('A1').options(pd.DataFrame, expand='table').value
       a    b
0.0  1.0  2.0
1.0  3.0  4.0
```

Matplotlib Excel

```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> plt.plot([1, 2, 3, 4, 5])
[<matplotlib.lines.Line2D at 0x1071706a0>]
>>> sht.pictures.add(fig, name='MyPlot', update=True)
<Picture 'MyPlot' in <Sheet [Workbook4]Sheet1>>
```

3.3 3. Macros: Call Python from Excel

`Run` (v0.16) Python VBA `RunPython`

`Run` Python `main` `xlsx`

Python `RunPython` :

```
Sub HelloWorld()
    RunPython "import hello; hello.world()"
End Sub
```

: Per default, `RunPython` expects `hello.py` in the same directory as the Excel file with the same name, **but you can change both of these things**: if your Python file is an a different folder, add that folder to the PYTHONPATH in the config. If the file has a different name, change the RunPython command accordingly.

Refer to the calling Excel book by using `xw.Book.caller()`:

```
# hello.py
import numpy as np
import xlwings as xw

def world():
    wb = xw.Book.caller()
    wb.sheets[0].range('A1').value = 'Hello World!'
```

To make this run, you'll need to have the xlwings add-in installed or have the workbooks setup in the standalone mode. The easiest way to get everything set up is to use the xlwings command line client from either a command prompt on Windows or a terminal on Mac: `xlwings quickstart myproject`.

Add-in & Settings

3.4 4. UDFs: User Defined Functions (Windows only)

Python UDF

```
import xlwings as xw

@xw.func
def hello(name):
    return 'Hello {0}'.format(name)
```

UDFs Pandas DataFrame

```
import xlwings as xw
import pandas as pd

@xw.func
@xw.arg('x', pd.DataFrame)
def correl2(x):
    # x arrives as DataFrame
    return x.corr()
```

Import this function into Excel by clicking the import button of the xlwings add-in: For a step-by-step tutorial, see *User Defined Functions (UDFs)*.

CHAPTER 4

Excel

```
>>> import xlwings as xw
>>> xw.Range('A1').value = 'something'
```

4.1 Python Excel

xw.Book app app app xw.books app

```
>>> app = xw.App()  # or something like xw.apps[10559] for existing apps, get␣
↪the available PIDs via xw.apps.keys()
>>> app.books['Book1']
```

	xw.Book	xw.books
	`xw.Book()`	`xw.books.add()`
	`xw.Book('Book1')`	`xw.books['Book1']`
	`xw.Book(r'C:/path/to/file.xlsx')`	`xw.books.open(r'C:/path/to/file.xlsx')`

: When specifying file paths on Windows, you should either use raw strings by putting an `r` in front of the string or use double back-slashes like so: `C:\\path\\to\\file.xlsx`.

4.2 Excel Python(RunPython)

VBA `RunPython` `xw.Book.caller()` *"RunPython" Python* Pyhton Excel `xw.Book.caller()`

4.3 (UDFs)

Unlike `RunPython`, UDFs don't need a call to `xw.Book.caller()`, see *User Defined Functions (UDFs)*. You'll usually use the `caller` argument which returns the xlwings range object from where you call the function.

CHAPTER 5

xlwings VBA

```
>>> import xlwings as xw
```

5.1

```
# Active app (i.e. Excel instance)
>>> app = xw.apps.active

# Active book
>>> wb = xw.books.active  # in active app
>>> wb = app.books.active  # in specific app

# Active sheet
>>> sht = xw.sheets.active  # in active book
>>> sht = wb.sheets.active  # in specific book

# Range on active sheet
>>> xw.Range('A1')  # on active sheet of active book of active app
```

A1 1 2

```
xw.Range('A1')
xw.Range('A1:C3')
```

( )

( )

```
xw.Range((1,1))
xw.Range((1,1), (3,3))
xw.Range('NamedRange')
xw.Range(xw.Range('A1'), xw.Range('B2'))
```

5.2

Excel 1 Python 0 :

```
xw.apps[763].books[0].sheets[0].range('A1')
xw.apps(10559).books(1).sheets(1).range('A1')
xw.apps[763].books['Book1'].sheets['Sheet1'].range('A1')
xw.apps(10559).books('Book1').sheets('Sheet1').range('A1')
```

app ID PID `xw.apps.keys()` PID

5.3 /

```
>>> rng = xw.Book().sheets[0].range('A1:D5')
>>> rng[0, 0]
 <Range [Workbook1]Sheet1!$A$1>
>>> rng[1]
 <Range [Workbook1]Sheet1!$B$1>
>>> rng[:, 3:]
<Range [Workbook1]Sheet1!$D$1:$D$5>
>>> rng[1:3, 1:3]
<Range [Workbook1]Sheet1!$B$2:$C$3>
```

5.4

/ `sheet.range` `sheet.cells`

```
>>> sht = xw.Book().sheets['Sheet1']
>>> sht['A1']
<Range [Book1]Sheet1!$A$1>
>>> sht['A1:B5']
<Range [Book1]Sheet1!$A$1:$B$5>
>>> sht[0, 1]
<Range [Book1]Sheet1!$B$1>
```

( )

( )

```
>>> sht[:10, :10]
<Range [Book1]Sheet1!$A$1:$J$10>
```

5.5

xlwings app range range app

```
>>> rng = xw.apps[10559].books[0].sheets[0].range('A1')
>>> rng.sheet.book.app
<Excel App 10559>
```

CHAPTER 6

xlwings options

```
>>> import xlwings as xw
```

6.1

python float, unicode, None datetime :

```
>>> import datetime as dt
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = 1
>>> sht.range('A1').value
1.0
>>> sht.range('A2').value = 'Hello'
>>> sht.range('A2').value
'Hello'
>>> sht.range('A3').value is None
True
>>> sht.range('A4').value = dt.datetime(2000, 1, 1)
>>> sht.range('A4').value
datetime.datetime(2000, 1, 1, 0, 0)
```

6.2

- Excel Python Python

```
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = [[1],[2],[3],[4],[5]]  # Column orientation␣
↪(nested list)
>>> sht.range('A1:A5').value
[1.0, 2.0, 3.0, 4.0, 5.0]
>>> sht.range('A1').value = [1, 2, 3, 4, 5]
>>> sht.range('A1:E1').value
[1.0, 2.0, 3.0, 4.0, 5.0]
```

:

```
>>> sht.range('A1').options(ndim=1).value
[1.0]
```

: Excel transpose: `sht.range('A1').options(transpose=True).value = [1,2,3,4]`

- ndim

```
>>> sht.range('A1:A5').options(ndim=2).value
[[1.0], [2.0], [3.0], [4.0], [5.0]]
>>> sht.range('A1:E1').options(ndim=2).value
[[1.0, 2.0, 3.0, 4.0, 5.0]]
```

- Excel Python

```
>>> sht.range('A10').value = [['Foo 1', 'Foo 2', 'Foo 3'], [10, 20, 30]]
>>> sht.range((10,1),(11,3)).value
[['Foo 1', 'Foo 2', 'Foo 3'], [10.0, 20.0, 30.0]]
```

: Excel `sht.range('A1').value = [[1,2],[3,4]]` `sht.range('A1').value = [1, 2]` `sht.range('A2').value = [3, 4]`

6.3

`expand` `options` `expand` `expand` , `options`

```
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = [[1,2], [3,4]]
```

( )

( )

```
>>> rng1 = sht.range('A1').expand('table')  # or just .expand()
>>> rng2 = sht.range('A1').options(expand='table')
>>> rng1.value
[[1.0, 2.0], [3.0, 4.0]]
>>> rng2.value
[[1.0, 2.0], [3.0, 4.0]]
>>> sht.range('A3').value = [5, 6]
>>> rng1.value
[[1.0, 2.0], [3.0, 4.0]]
>>> rng2.value
[[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]
```

'table' 'down' ( ) 'right' ( )

: Using `expand()` together with a named Range as top left cell gives you a flexible setup in Excel: You can move around the table and change its size without having to adjust your code, e.g. by using something like `sht.range('NamedRange').expand().value`.

6.4 NumPy

NumPy nan None options convert=np.array

```
>>> import numpy as np
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = np.eye(3)
>>> sht.range('A1').options(np.array, expand='table').value
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

6.5 Pandas (DataFrame)

```
>>> sht = xw.Book().sheets[0]
>>> df = pd.DataFrame([[1.1, 2.2], [3.3, None]], columns=['one', 'two'])
>>> df
   one  two
0  1.1  2.2
1  3.3  NaN
>>> sht.range('A1').value = df
>>> sht.range('A1:C3').options(pd.DataFrame).value
   one  two
```

( )

( )

```
0  1.1  2.2
1  3.3  NaN
# options: work for reading and writing
>>> sht.range('A5').options(index=False).value = df
>>> sht.range('A9').options(index=False, header=False).value = df
```

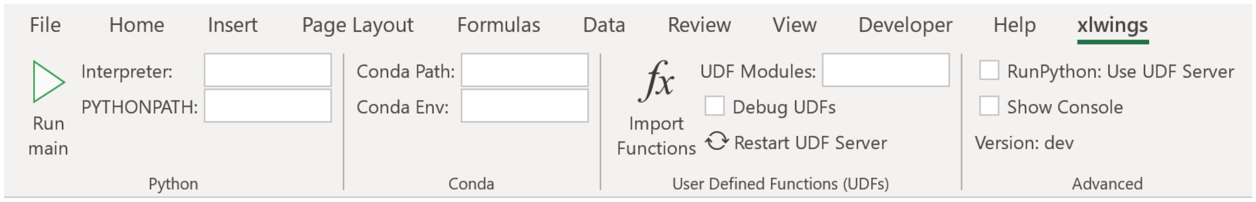
6.6 Pandas (Serie)

```
>>> import pandas as pd
>>> import numpy as np
>>> sht = xw.Book().sheets[0]
>>> s = pd.Series([1.1, 3.3, 5., np.nan, 6., 8.], name='myseries')
>>> s
0    1.1
1    3.3
2    5.0
3    NaN
4    6.0
5    8.0
Name: myseries, dtype: float64
>>> sht.range('A1').value = s
>>> sht.range('A1:B7').options(pd.Series).value
0    1.1
1    3.3
2    5.0
3    NaN
4    6.0
5    8.0
Name: myseries, dtype: float64
```

: Excel NumPy Pandas `sht.range('A1').value = np.eye(10)`

CHAPTER 7

Add-in & Settings



The xlwings add-in is the preferred way to be able to use the `Run main` button, `RunPython` or `UDFs`. Note that you don't need an add-in if you just want to manipulate Excel by running a Python script.

: The ribbon of the add-in is compatible with Excel >= 2007 on Windows and >= 2016 on Mac. On Mac, all UDF related functionality is not available.

: The add-in is password protected with the password `xlwings`. For debugging or to add new extensions, you need to unprotect it. Alternatively, you can also install the add-in via `xlwings addin install --unprotected`.

7.1 main

0.16.0 .

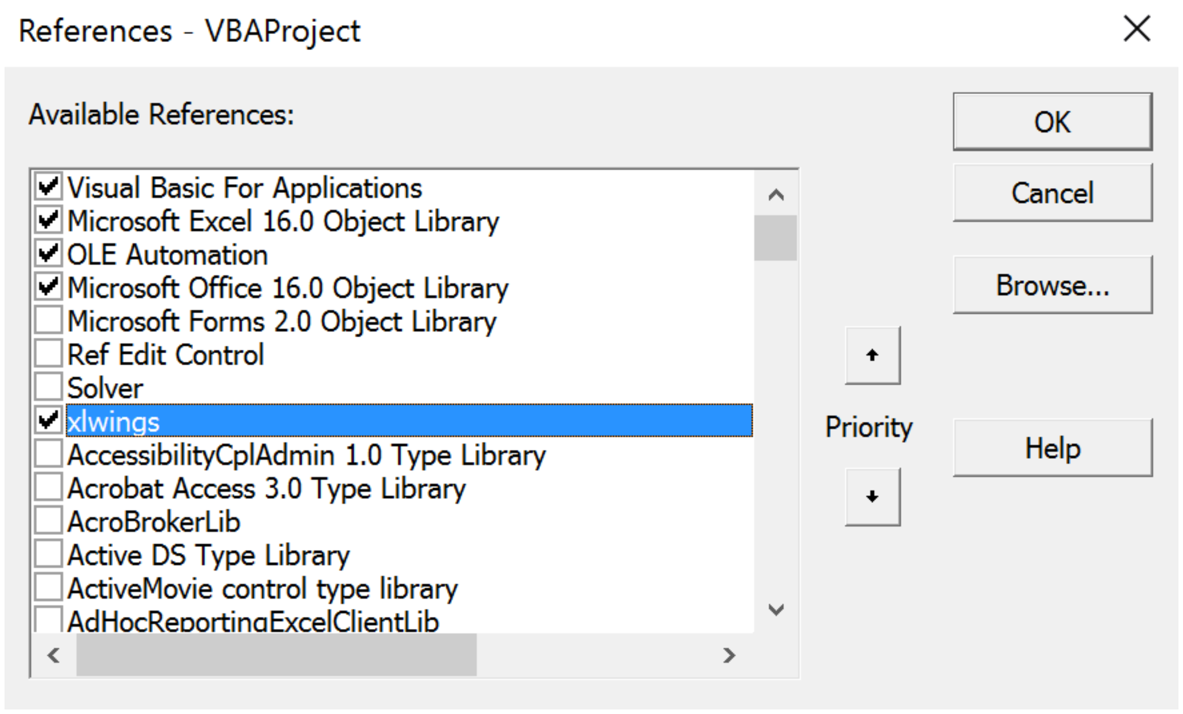
The `Run main` button is the easiest way to run your Python code: It runs a function called `main` in a Python module that has the same name as your workbook. This allows you to save your workbook as `xlsx` without enabling macros. The `xlwings quickstart` command will create a workbook that will automatically work with the `Run` button.

7.2

To install the add-in, use the command line client:

```
xlwings addin install
```

Technically, this copies the add-in from Python's installation directory to Excel's `XLSTART` folder. Then, to use `RunPython` or `UDFs` in a workbook, you need to set a reference to `xlwings` in the VBA editor, see screenshot (Windows: `Tools > References...`, Mac: it's on the lower left corner of the VBA editor). Note that when you create a workbook via `xlwings quickstart`, the reference should already be set.



- ✓ Visual Basic For Applications
- ✓ Microsoft Excel 16.0 Object Library
- ✓ OLE Automation
- ✓ Microsoft Office 16.0 Object Library
- ☐ Microsoft Forms 2.0 Object Library
- ☐ Ref Edit Control
- ☐ Solver
- ✓ xlwings
- ☐ AccessibilityCplAdmin 1.0 Type Library
- ☐ Acrobat Access 3.0 Type Library
- ☐ AcroBrokerLib
- ☐ Active DS Type Library
- ☐ ActiveMovie control type library
- ☐ AdHocReportingExcelClientLib

7.3 User Settings

When you install the add-in for the first time, it will get auto-configured and therefore, a `quickstart` project should work out of the box. For fine-tuning, here are the available settings:

- `Interpreter`: This is the path to the Python interpreter. This works also with virtual or conda envs on Mac. If you use conda envs on Windows, then leave this empty and use `Conda Path` and `Conda Env` below instead. Examples: `"C:\Python39\pythonw.exe"` or `"/usr/local/bin/python3.9"`. Note that in the settings, this is stored as `Interpreter_Win` or `Interpreter_Mac`, respectively, see below!

- `PYTHONPATH`: If the source file of your code is not found, add the path to its directory here.
- `Conda Path`: Windows conda Anaconda Miniconda `C:\Users\Username\Miniconda3` `%USERPROFILE%\Anaconda` conda 4.6
- `Conda Env`: If you are on Windows and use Anaconda or Miniconda, type here the name of your conda env, e.g. `base` for the base installation or `myenv` for a conda env with the name `myenv`.
- `UDF Modules` : UDF Python (.py) ";" `UDF_MODULES = "common_udfs; myproject"` Excel `.py`
- `Debug UDFs`: xlwings COM .
- `RunPython: Use UDF Server`: UDF COM Python
- `Restart UDF Server`: This restarts the UDF Server/Python interpreter.
- `Show Console`: Check the box in the ribbon or set the config to `TRUE` if you want the command prompt to pop up. This currently only works on Windows.

7.3.1 Anaconda/Miniconda

If you use Anaconda or Miniconda on Windows, you will need to set your `Conda Path` and `Conda Env` settings, as you will otherwise get errors when using `NumPy` etc. In return, leave `Interpreter` empty.

7.4 Environment Variables

With environment variables, you can set dynamic paths e.g. to your interpreter or `PYTHONPATH`:

- On Windows, you can use all environment variables like so: `%USERPROFILE%\Anaconda`.
- On macOS, the following special variables are supported: `$HOME`, `$APPLICATIONS`, `$DOCUMENTS`, `$DESKTOP`.

7.5 User Config: Ribbon/Config File

xlwings

- Windows: `.xlwings\xlwings.conf` in your home folder, that is usually `C:\Users\<username>`
- macOS: `~/Library/Containers/com.microsoft.Excel/Data/xlwings.conf`

The format is as follows (currently the keys are required to be all caps) - note the OS specific Interpreter settings!

```
"INTERPRETER_WIN","C:\path\to\python.exe"
"INTERPRETER_MAC","/path/to/python"
"PYTHONPATH",""
"CONDA PATH",""
"CONDA ENV",""
"UDF MODULES",""
"DEBUG UDFS",""
"USE UDF SERVER",""
"SHOW CONSOLE",""
"ONEDRIVE",""
```

: The `ONEDRIVE` setting has to be edited directly in the file, there is currently no possibility to edit it via the ribbon. Usually, it is only required if you are either on macOS or if your environment vars on Windows are not correctly set or if you have a private and corporate location and don't want to go with the default one. `ONEDRIVE` has to point to the root folder of your local OneDrive folder.

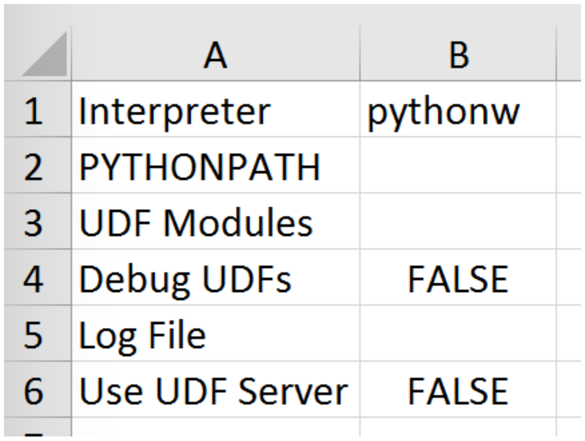
7.6

`xlwings.conf`

7.7 xlwings.conf

`xlwings.conf` `xlwings quickstart`

`xlwings.conf`



	A	B
1	Interpreter	pythonw
2	PYTHONPATH	
3	UDF Modules	
4	Debug UDFs	FALSE
5	Log File	
6	Use UDF Server	FALSE

7.8 VBA

Sometimes, it might be useful to run xlwings code without having to install an add-in first. To do so, you need to use the `standalone` option when creating a new project: `xlwings quickstart myproject --standalone`.

This will add the content of the add-in as a single VBA module so you don't need to set a reference to the add-in anymore. It will also include `Dictionary.cls` as this is required on macOS. It will still read in the settings from your `xlwings.conf` if you don't override them by using a sheet with the name `xlwings.conf`.

CHAPTER 8

RunPython

8.1 xlwings

`Run main` (v0.16) `RunPython` VBA xlwings (VBA) *Add-in & Settings*.

quickstart *Command Line Client (CLI)*

```
$ xlwings quickstart myproject
```

8.2 "RunPython" Python

In the VBA Editor (`Alt-F11`), write the code below into a VBA module. `xlwings quickstart` automatically adds a new module with a sample call. If you rather want to start from scratch, you can add a new module via `Insert > Module`.

```
Sub HelloWorld()
    RunPython ("import hello; hello.world()")
End Sub
```

`hello.py`

```
# hello.py
import numpy as np
import xlwings as xw

def world():
```

( )

( )

```
    wb = xw.Book.caller()
    wb.sheets[0].range('A1').value = 'Hello World!'
```

HelloWorld VBA F5

: Place `xw.Book.caller()` within the function that is being called from Excel and not outside as global variable. Otherwise it prevents Excel from shutting down properly upon exiting and leaves you with a zombie process when you use `Use UDF Server = True`.

8.3

While it's technically possible to include arguments in the function call within `RunPython`, it's not very convenient. Also, `RunPython` does not allow you to return values. To overcome these issues, use UDFs, see *User Defined Functions (UDFs)* - however, this is currently limited to Windows only.

CHAPTER 9

User Defined Functions (UDFs)

:

- Windows (UDF)
- .
- API : *UDF* .

9.1 Excel

1) Enable `Trust access to the VBA project object model` under `File > Options > Trust Center > Trust Center Settings > Macro Settings`

2) Install the add-in via command prompt: `xlwings addin install` (see *Add-in & Settings*).

9.2

`xlwings quickstart myproject` (*Command Line Client (CLI)*) xlwings

9.3 UDF

"quickstart" Python

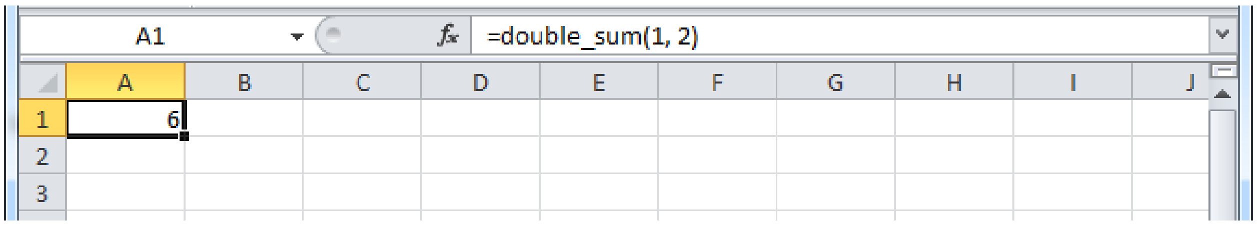
-
- .py .xlsm

xlwings UDF Modules

myproject.xlsm myproject.py:

```python
import xlwings as xw

@xw.func
def double_sum(x, y):
    """Returns twice the sum of the two arguments"""
    return 2 * (x + y)
```

- xlwings Import Python UDFs myproject.py
- =double_sum(1, 2)



A1 =double_sum(1, 2)

	A	B	C	D	E	F	G	H	I	J
1	6									
2										
3										

- ( ) Excel

:

-
- (Ctrl-Alt-F9) Python Restart UDF Server [Excel2013]
- @xw.func Excel xlwings xlwings VBA Python

9.4

Excel

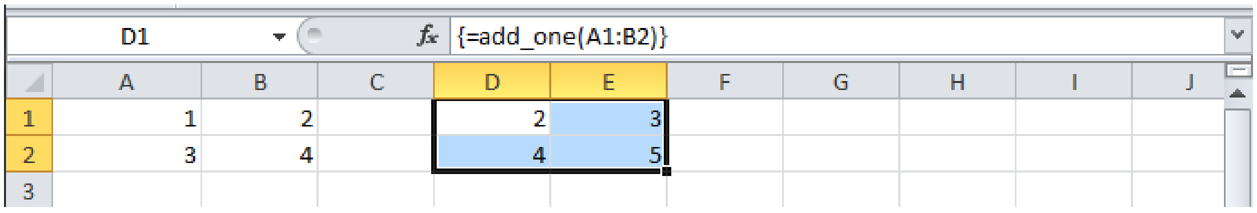
Python

1:

```
@xw.func
def add_one(data):
    return [[cell + 1 for cell in row] for row in data]
```

Excel

- Import Python UDFs
- A1:B2
- D1:E2
- =add_one(A1:B2)
- Ctrl+Shift+Enter



D1 {=add_one(A1:B2)}

	A	B	C	D	E	F	G	H	I	J
1	1	2		2	3					
2	3	4		4	5					
3										

9.4.1 : ndim

[[1, 2], [3, 4]] “2 ” : 'float'

Excel / 2 2 :

```
@xw.func
@xw.arg('data', ndim=2)
def add_one(data):
    return [[cell + 1 for cell in row] for row in data]
```

9.5 NumPy Pandas

UDF NumPy array Pandas DataFrame Python

numpy array :

```
import xlwings as xw
import numpy as np

@xw.func
@xw.arg('x', np.array, ndim=2)
@xw.arg('y', np.array, ndim=2)
def matrix_mult(x, y):
    return x @ y
```

: Python >= 3.5 NumPy >= 1.10 `x.dot(y)` `x @ y`

Pandas `CORREL` Excel `CORREL` 2 Pandas `CORREL2` :

```python
import xlwings as xw
import pandas as pd

@xw.func
@xw.arg('x', pd.DataFrame, index=False, header=False)
@xw.ret(index=False, header=False)
def CORREL2(x):
    """Like CORREL, but as array formula for more than 2 data sets"""
    return x.corr()
```

9.6 @xw.arg @xw.ret

UDF `options` `Range` (`@xw.arg`) (`@xw.ret`) x pandas DataFrame :

```python
@xw.func
@xw.arg('x', pd.DataFrame)
@xw.ret(index=False)
def myfunction(x):
   # x is a DataFrame, do something with it
   return x
```

9.7

: Excel `expand` `=UNIQUE()` 2018 9 Office 365 Insider Fast

Excel `Ctrl-Shift-Enter` v0.10 xlwings UDF

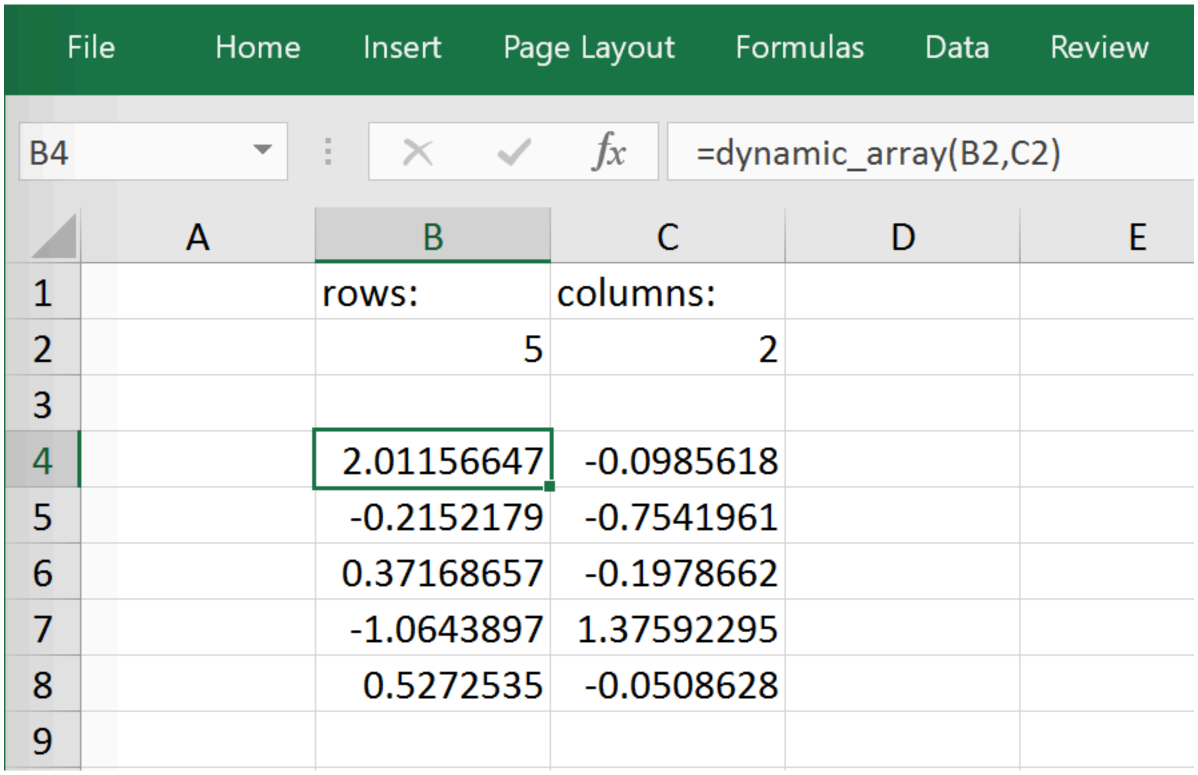
UDF

```python
import numpy as np

@xw.func
@xw.ret(expand='table')
```

( )

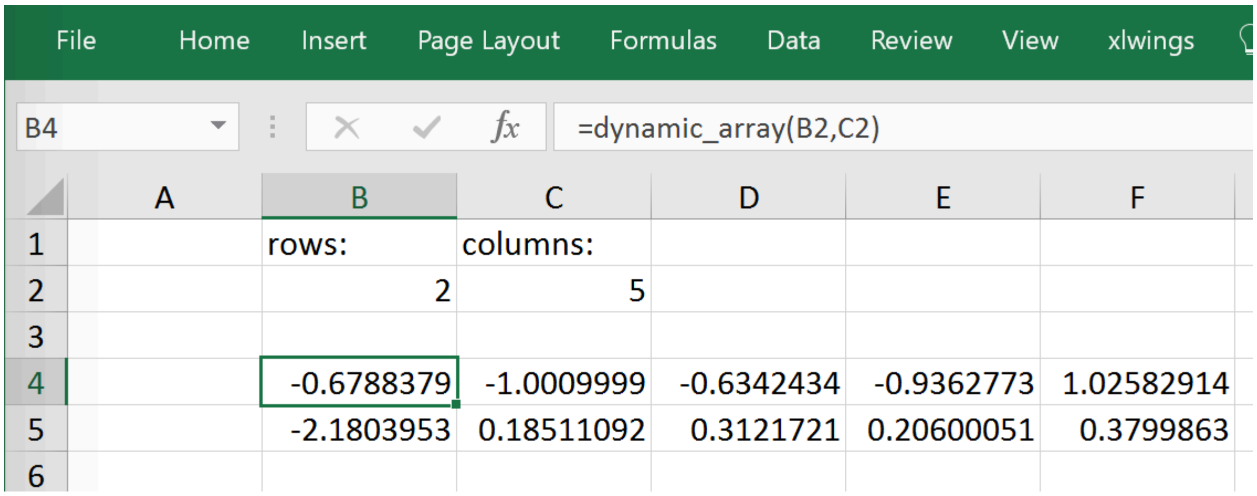
( )

```python
def dynamic_array(r, c):
    return np.random.randn(int(r), int(c))
```



B4 =dynamic_array(B2,C2)

	A	B	C	D	E
1		rows:	columns:		
2		5	2		
3					
4		2.01156647	-0.0985618		
5		-0.2152179	-0.7541961		
6		0.37168657	-0.1978662		
7		-1.0643897	1.37592295		
8		0.5272535	-0.0508628		
9					



B4 =dynamic_array(B2,C2)

	A	B	C	D	E	F
1		rows:	columns:			
2		2	5			
3						
4		-0.6788379	-1.0009999	-0.6342434	-0.9362773	1.02582914
5		-2.1803953	0.18511092	0.3121721	0.20600051	0.3799863
6						

:

- ;
- v0.15.0 `=TODAY()` v0.15.0 UDF 1 ;
- v0.15.0 xlwings >= v0.15.0 `Ctrl-Shift-Enter`

9.8

x y Excel

```python
import xlwings as xw

@xw.func
@xw.arg('x', doc='This is x.')
@xw.arg('y', doc='This is y.')
def double_sum(x, y):
    """Returns twice the sum of the two arguments"""
    return 2 * (x + y)
```

9.9 The "caller" argument

You often need to know which cell called the UDF. For this, xlwings offers the reserved argument `caller` which returns the calling cell as xlwings range object:

```python
@xw.func
def get_caller_address(caller):
    # caller will not be exposed in Excel, so use it like so:
    # =get_caller_address()
    return caller.address
```

Note that `caller` will not be exposed in Excel but will be provided by xlwings behind the scenes.

9.10 "vba"

By using the `vba` keyword, you can get access to any Excel VBA object in the form of a pywin32 object. For example, if you wanted to pass the sheet object in the form of its `CodeName`, you can do it as follows:

```python
@xw.func
@xw.arg('sheet1', vba='Sheet1')
def get_name(sheet1):
    # call this function in Excel with:
    # =get_name()
    return sheet1.Name
```

Note that `vba` arguments are not exposed in the UDF but automatically provided by xlwings.

9.11

On Windows, as an alternative to calling macros via *RunPython*, you can also use the `@xw.sub` decorator:

```python
import xlwings as xw

@xw.sub
def my_macro():
    """Writes the name of the Workbook into Range("A1") of Sheet 1"""
    wb = xw.Book.caller()
    wb.sheets[0].range('A1').value = wb.name
```

After clicking on `Import Python UDFs`, you can then use this macro by executing it via `Alt + F8` or by binding it e.g. to a button. To do the latter, make sure you have the `Developer` tab selected under `File > Options > Customize Ribbon`. Then, under the `Developer` tab, you can insert a button via `Insert > Form Controls`. After drawing the button, you will be prompted to assign a macro to it and you can select `my_macro`.

9.12 VBA UDF

VBA 2 :

```vb
Sub MySub()

Dim arr() As Variant
Dim i As Long, j As Long

    arr = my_imported_function(...)

    For j = LBound(arr, 2) To UBound(arr, 2)
        For i = LBound(arr, 1) To UBound(arr, 1)
            Debug.Print "(" & i & "," & j & ")", arr(i, j)
        Next i
    Next j

End Sub
```

9.13 UDF

: This is an experimental feature

v0.14.0 .

xlwings Excel #N/A waiting... Excel

async_mode='threading' API I/O

I/O time.sleep :

```
import xlwings as xw
import time

@xw.func(async_mode='threading')
def myfunction(a):
    time.sleep(5)  # long running tasks
    return a
```

You can use this function like any other xlwings function, simply by putting =myfunction("abcd") into a cell (after you have imported the function, of course).

xlwings Excel 2010 xlwings Excel

CHAPTER 10

Matplotlib & Plotly Charts

10.1 Matplotlib

pictures.add() Matplotlib Excel

10.1.1

:

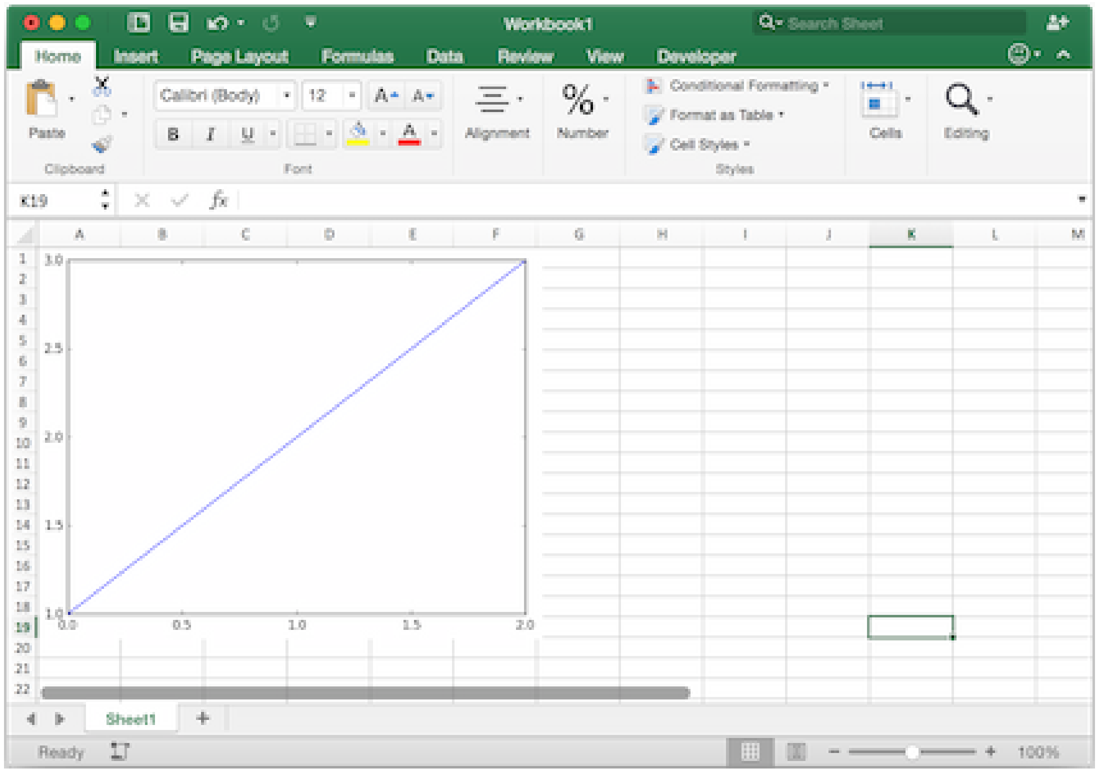
```
>>> import matplotlib.pyplot as plt
>>> import xlwings as xw

>>> fig = plt.figure()
>>> plt.plot([1, 2, 3])

>>> sht = xw.Book().sheets[0]
>>> sht.pictures.add(fig, name='MyPlot', update=True)
```

: `update=True`, Excel `pictures.add()` (`'MyPlot'`)

10.1.2 Excel

RunPython

Windows *UDF* :



```
@xw.func
def myplot(n, caller):
    fig = plt.figure()
    plt.plot(range(int(n)))
    caller.sheet.pictures.add(fig, name='MyPlot', update=True)
    return 'Plotted with n={}'.format(n)
```

UDF   B2   B1

10.1.3

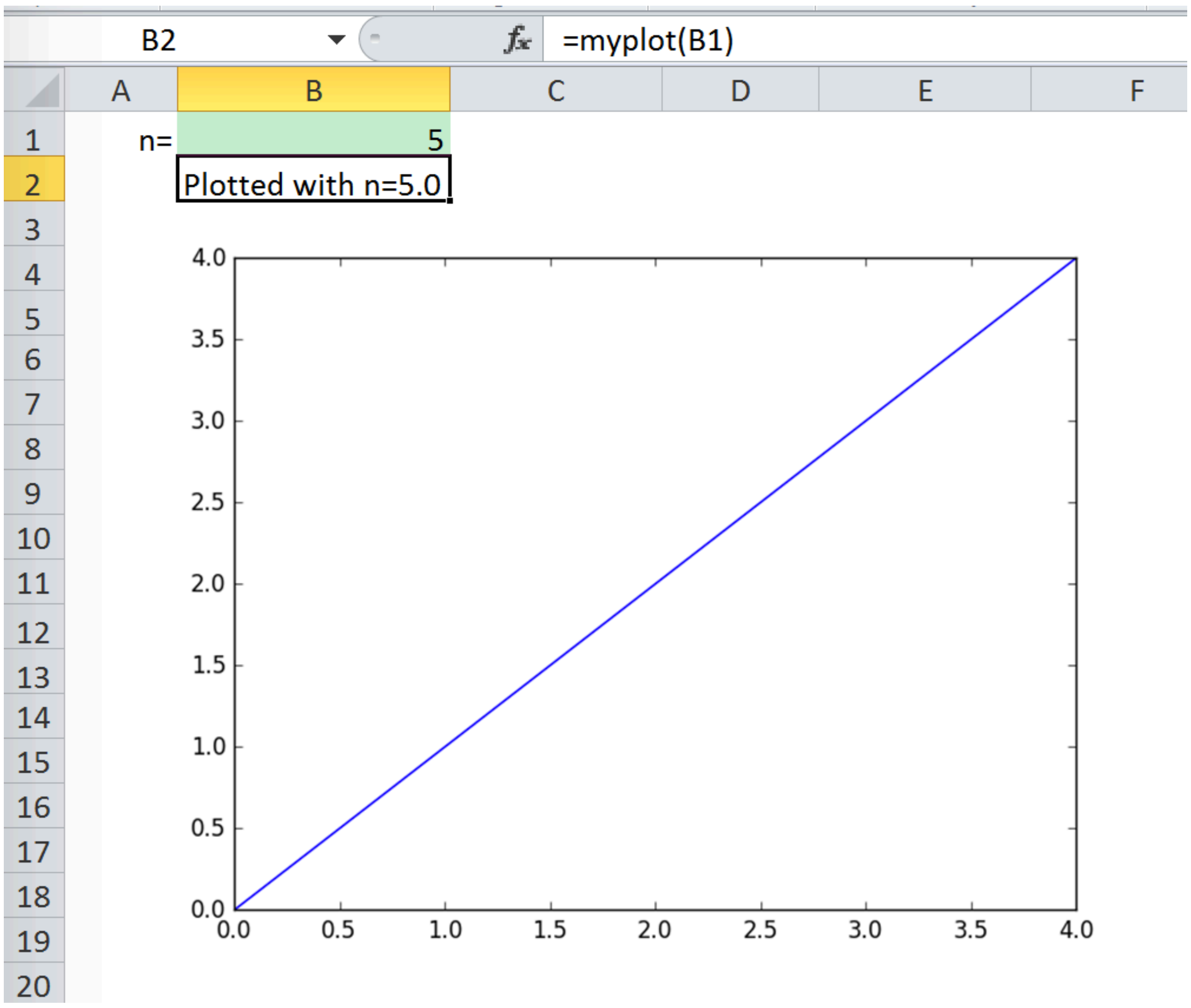
pictures.add()   *xlwings.Picture()*.

:

```
>>> sht = xw.Book().sheets[0]
>>> sht.pictures.add(fig, name='MyPlot', update=True,
                     left=sht.range('B5').left, top=sht.range('B5').top)
```

:

```
>>> plot = sht.pictures.add(fig, name='MyPlot', update=True)
>>> plot.height /= 2
>>> plot.width /= 2
```


B2 =myplot(B1)

n= 5

Plotted with n=5.0

10.1.4 Matplotlib

matplotlib figure

- PyPlot :

```
import matplotlib.pyplot as plt
fig = plt.figure()
plt.plot([1, 2, 3, 4, 5])
```

:

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4, 5])
fig = plt.gcf()
```

- :

```
from matplotlib.figure import Figure
fig = Figure(figsize=(8, 6))
ax = fig.add_subplot(111)
ax.plot([1, 2, 3, 4, 5])
```

- Pandas:

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])
ax = df.plot(kind='bar')
fig = ax.get_figure()
```

10.2 Plotly static charts

This feature requires xlwings *PRO*.

10.2.1 Prerequisites

In addition to `plotly` you will need `orca`. The easiest way to get it is via conda:

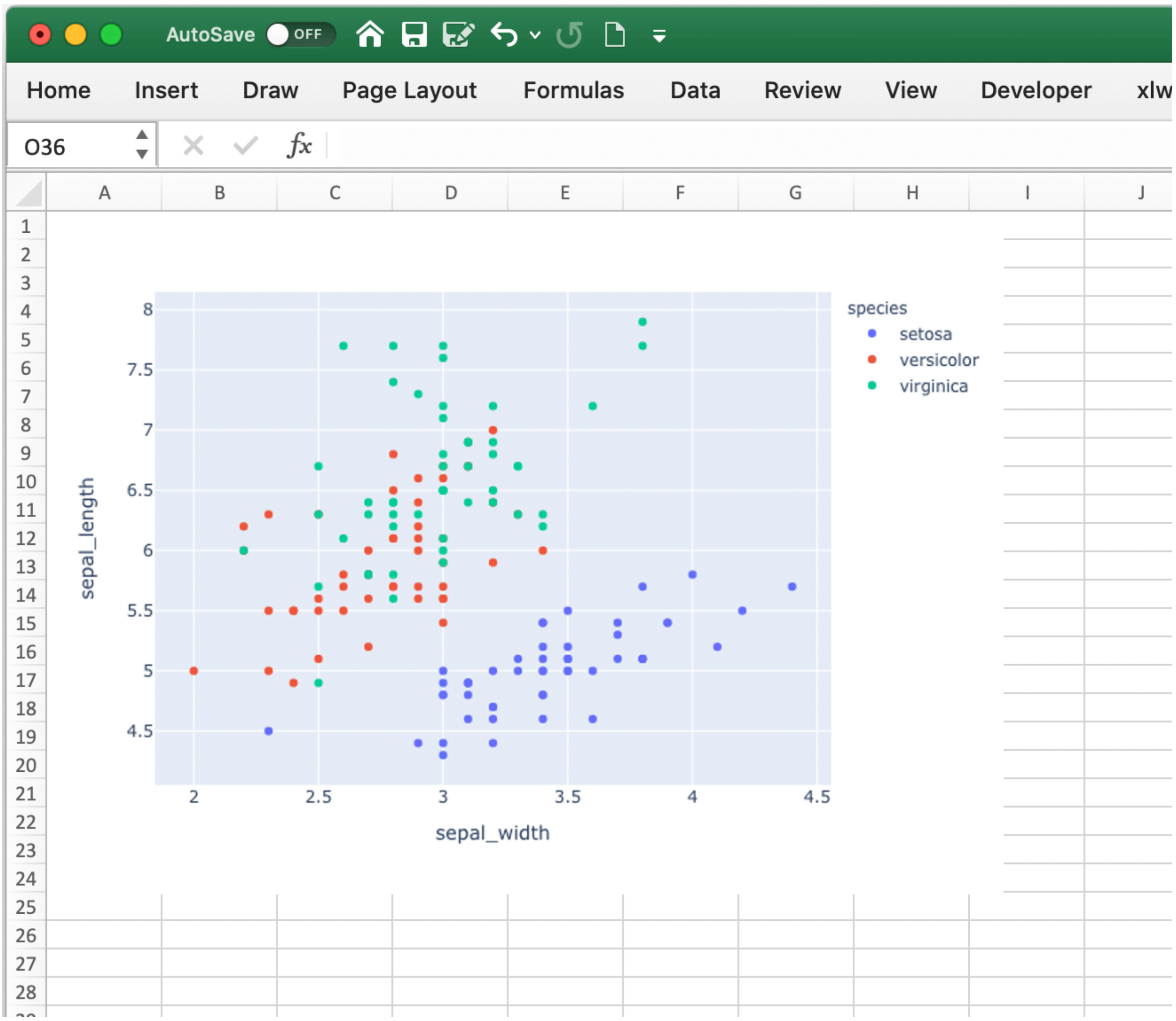
```
$ conda install -c plotly plotly-orca psutil requests
```

For alternative ways of installation, see: https://plotly.com/python/static-image-export/

10.2.2 How to use

It works the same as with Matplotlib, however, rendering a Plotly chart takes slightly longer. Here is a sample:

```python
import xlwings as xw
import plotly.express as px

# Plotly chart
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species")

# Add it to Excel
wb = xw.Book()
wb.sheets[0].pictures.add(fig, name='IrisScatterPlot', update=True)
```



CHAPTER 11

Jupyter Notebooks: Interact with Excel

When you work with Jupyter notebooks, you may use Excel as an interactive data viewer or scratchpad from where you can load DataFrames. The two convenience functions *view* and *load* make this really easy.

: The *view* and *load* functions should exclusively be used for interactive work. If you write scripts, use the xlwings API as introduced under and .

11.1 The view function

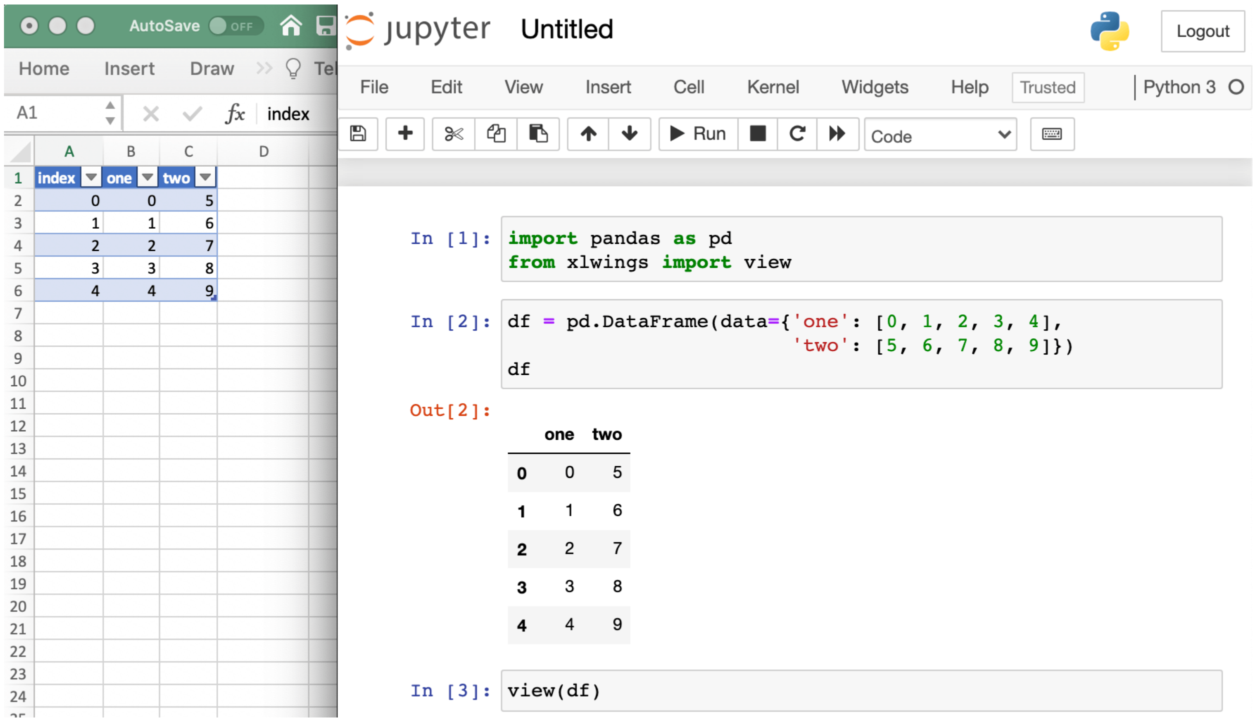
The view function accepts pretty much any object of interest, whether that's a number, a string, a nested list or a NumPy array or a pandas DataFrame. By default, it writes the data into an Excel table in a new workbook. If you wanted to reuse the same workbook, provide a `sheet` object, e.g. `view(df, sheet=xw.sheets.active)`, for further options see *view*.

0.22.0 : Earlier versions were not formatting the output as Excel table

11.2 The load function

To load in a range in an Excel sheet as pandas DataFrame, use the `load` function. If you only select one cell, it will auto-expand to cover the whole range. If, however, you select a specific range that is bigger than one cell, it will load in only the selected cells. If the data in Excel does not have an index or header, set them to `False` like this: `xw.load(index=False)`, see also *load*.

0.22.0 .

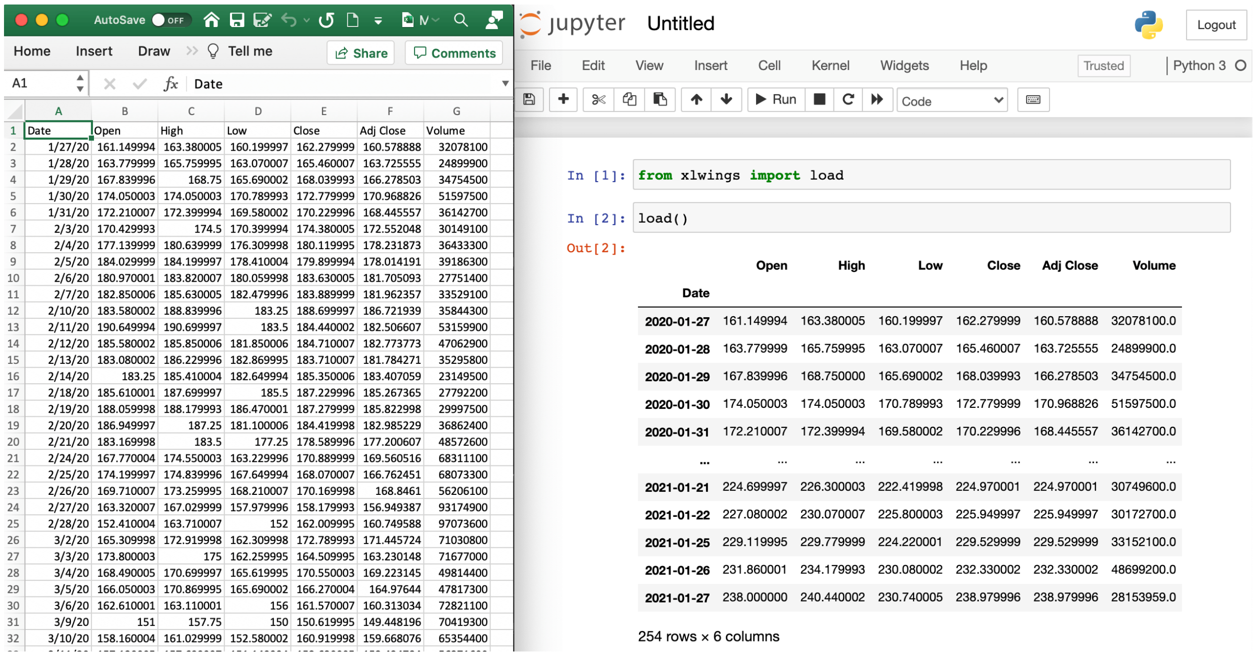


	A	B	C	D
1	index	one	two	
2	0	0	5	
3	1	1	6	
4	2	2	7	
5	3	3	8	
6	4	4	9	

```
In [1]: import pandas as pd
        from xlwings import view

In [2]: df = pd.DataFrame(data={'one': [0, 1, 2, 3, 4],
                                'two': [5, 6, 7, 8, 9]})
        df

Out[2]:
```

	one	two
0	0	5
1	1	6
2	2	7
3	3	8
4	4	9

```
In [3]: view(df)
```



	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	1/27/20	161.149994	163.380005	160.199997	162.279999	160.578888	32078100
3	1/28/20	163.779999	165.759995	163.070007	165.460007	163.725555	24899900
4	1/29/20	167.839996	168.75	165.690002	168.039993	166.278503	34754500
5	1/30/20	174.050003	174.050003	170.789993	172.779999	170.968826	51597500
6	1/31/20	172.210007	172.399994	169.580002	170.229996	168.445557	36142700
7	2/3/20	170.429993	174.5	170.399994	174.380005	172.552048	30149100
8	2/4/20	177.139999	180.639999	176.309998	180.119995	178.231873	36433300
9	2/5/20	184.029999	184.199997	178.410004	179.899994	178.014191	39186300
10	2/6/20	180.970001	183.820007	180.059998	183.630005	181.705093	27751400
11	2/7/20	182.850006	185.630005	182.479996	183.889999	181.962357	33529100
12	2/10/20	183.580002	188.839996	183.25	188.699997	186.721939	35844300
13	2/11/20	190.649994	190.699997	183.5	184.440002	182.506607	53159900
14	2/12/20	185.580002	185.850006	181.850006	184.710007	182.773773	47062900
15	2/13/20	183.080002	186.229996	182.869995	183.710007	181.784271	35295800
16	2/14/20	183.25	185.410004	182.649994	185.350006	183.407059	23149500
17	2/18/20	185.610001	187.699997	185.5	187.229996	185.267365	27792200
18	2/19/20	188.059998	188.179993	186.470001	187.279999	185.822998	29997500
19	2/20/20	186.949997	187.25	181.100006	184.419998	182.985229	36862400
20	2/21/20	183.169998	183.5	177.25	178.589996	177.200607	48572600
21	2/24/20	167.770004	174.550003	163.229996	170.889999	169.560516	68311100
22	2/25/20	174.199997	174.839996	167.649994	168.070007	166.762451	68073300
23	2/26/20	169.710007	173.259995	168.210007	170.169998	168.8461	56206100
24	2/27/20	163.320007	167.029999	157.979996	158.179993	156.949387	93174900
25	2/28/20	152.410004	163.710007	152	162.009995	160.749588	97073600
26	3/2/20	165.309998	172.919998	162.309998	172.789993	171.445724	71030800
27	3/3/20	173.800003	175	162.259995	164.509995	163.230148	71677000
28	3/4/20	168.490005	170.699997	165.619995	170.550003	169.223145	49814400
29	3/5/20	166.050003	170.869995	165.690002	166.270004	164.97644	47817300
30	3/6/20	162.610001	163.110001	156	161.570007	160.313034	72821100
31	3/9/20	151	157.75	150	150.619995	149.448196	70419300
32	3/10/20	158.160004	161.029999	152.580002	160.919998	159.668076	65354400

```
In [1]: from xlwings import load

In [2]: load()

Out[2]:
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2020-01-27	161.149994	163.380005	160.199997	162.279999	160.578888	32078100.0
2020-01-28	163.779999	165.759995	163.070007	165.460007	163.725555	24899900.0
2020-01-29	167.839996	168.750000	165.690002	168.039993	166.278503	34754500.0
2020-01-30	174.050003	174.050003	170.789993	172.779999	170.968826	51597500.0
2020-01-31	172.210007	172.399994	169.580002	170.229996	168.445557	36142700.0
...	...	...	...	...	...	...
2021-01-21	224.699997	226.300003	222.419998	224.970001	224.970001	30749600.0
2021-01-22	227.080002	230.070007	225.800003	225.949997	225.949997	30172700.0
2021-01-25	229.119995	229.779999	224.220001	229.529999	229.529999	33152100.0
2021-01-26	231.860001	234.179993	230.080002	232.330002	232.330002	48699200.0
2021-01-27	238.000000	240.440002	230.740005	238.979996	238.979996	28153959.0

254 rows × 6 columns

CHAPTER 12

Command Line Client (CLI)

xlwings comes with a command line client. On Windows, type the commands into a `Command Prompt`, on Mac, type them into a `Terminal`. To get an overview of all commands, simply type `xlwings` and hit Enter:

```
addin                 Run "xlwings addin install" to install the Excel add-
                      in (will be copied to the XLSTART folder). Instead of
                      "install" you can also use "update", "remove" or
                      "status". Note that this command may take a while. Use
                      the "--unprotected" flag to install the add-in without
                      password protection. You can install your custom add-
                      in by providing the name or path via the --file flag,
                      e.g. "xlwings add-in install --file custom.xlam"
                      (New in 0.6.0, the unprotected flag was added in 0.20.4)
quickstart            Run "xlwings quickstart myproject" to create a folder
                      called "myproject" in the current directory with an
                      Excel file and a Python file, ready to be used. Use
                      the "--standalone" flag to embed all VBA code in the
                      Excel file and make it work without the xlwings add-
                      in.
runpython             macOS only: run "xlwings runpython install" if you
                      want to enable the RunPython calls without installing
                      the add-in. This will create the following file:
                      ~/Library/Application
                      Scripts/com.microsoft.Excel/xlwings.applescript
                      (new in 0.7.0)
restapi               Use "xlwings restapi run" to run the xlwings REST API
                      via Flask dev server. Accepts "--host" and "--port" as
```

( )

( )

```
                    optional arguments.
license             xlwings PRO: Use "xlwings license update -k KEY" where
                    "KEY" is your personal (trial) license key. This will
                    update ~/.xlwings/xlwings.conf with the LICENSE_KEY
                    entry. If you have a paid license, you can run
                    "xlwings license deploy" to create a deploy key. This
                    is not available for trial keys.
config              Run "xlwings config create" to create the user config
                    file (~/.xlwings/xlwings.conf) which is where the
                    settings from the Ribbon add-in are stored. It will
                    configure the Python interpreter that you are running
                    this command with. To reset your configuration, run
                    this with the "--force" flag which will overwrite your
                    current configuration.
                    (New in 0.19.5)
code                Run "xlwings code embed" to embed all Python modules
                    of the current dir in your active Excel file. Use the
                    "--file" flag to only import a single file by
                    providing its path. To run embedded code, you need an
                    xlwings PRO license.
                    (New in 0.20.2)
```

CHAPTER 13

xlwings Reports

This feature requires xlwings *PRO*.

13.1 Quickstart

xlwings Reports is part of xlwings PRO and a solution for template-based Excel and PDF reporting. It allows business users without Python knowledge to create & maintain Excel templates without having to go back to a Python developer for every change: xlwings Reports separates the Python code (that gets and prepares all the data) from the Excel template (that defines which data goes where and how it should be formatted). See also the xlwings Reports homepage. You can render one sheet at the time via *mysheet.render_template* or use the higher-level convenience function `xw.create_report` which first copies the template workbook and then loops through all sheets.

13.1.1 Render Sheets

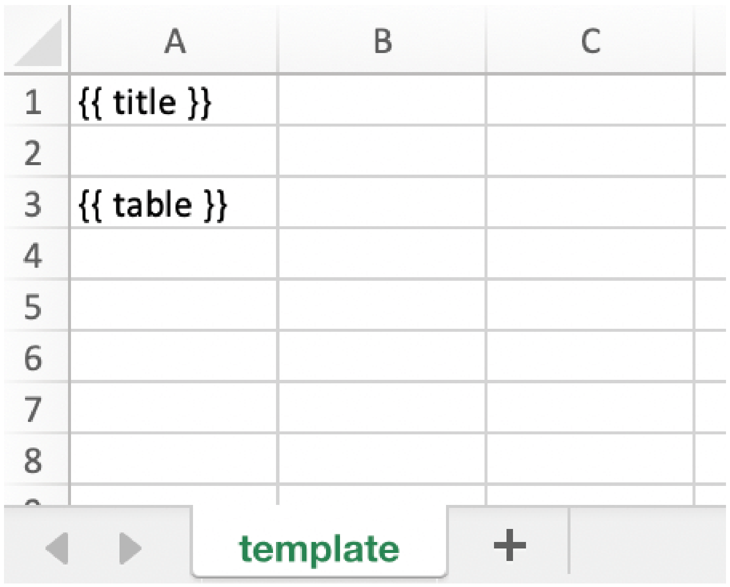
Let's first look at how to render a single sheet. This is a workbook stored as `Book1.xlsx`:

Running the following code:

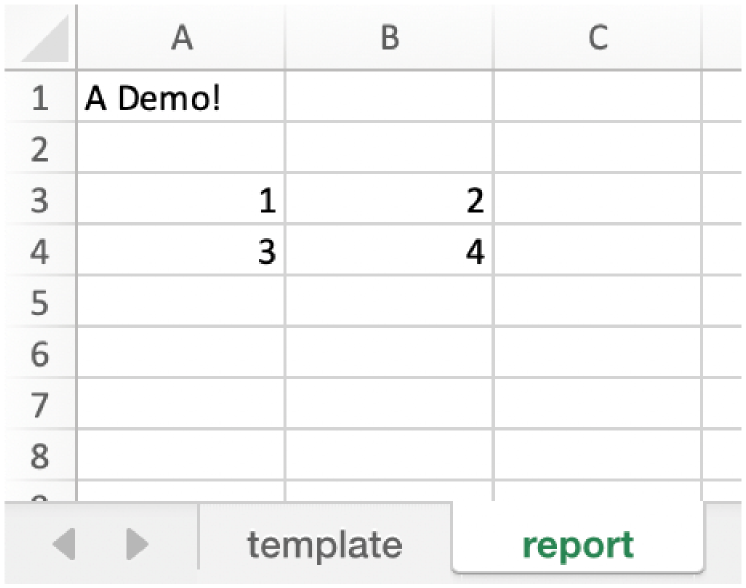
```python
import xlwings as xw
wb = xw.Book('Book1.xlsx')
sheet = wb.sheets['template'].copy(name='report')
sheet.render_template(title='A Demo!', table=[[1, 2], [3, 4]])
wb.to_pdf()  # requires xlwings >=0.21.1
```

Leaves you with this:

See also the *API reference*.



	A	B	C
1	{{ title }}		
2			
3	{{ table }}		
4			
5			
6			
7			
8			



	A	B	C
1	A Demo!		
2			
3	1	2	
4	3	4	
5			
6			
7			
8			

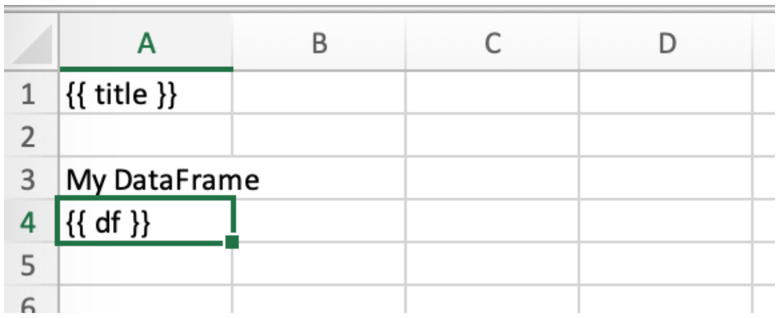
0.22.0 .

13.1.2 Render Workbooks

If your template is a full workbook, you can use the `create_report` function. Start by creating the following Python script `my_template.py`:

```python
from xlwings.pro.reports import create_report
import pandas as pd

df = pd.DataFrame(data=[[1,2],[3,4]])
wb = create_report('my_template.xlsx', 'my_report.xlsx', title='MyTitle', df=df)
wb.to_pdf()  # requires xlwings >=0.21.1
```

Then create the following Excel file called `my_template.xlsx`:

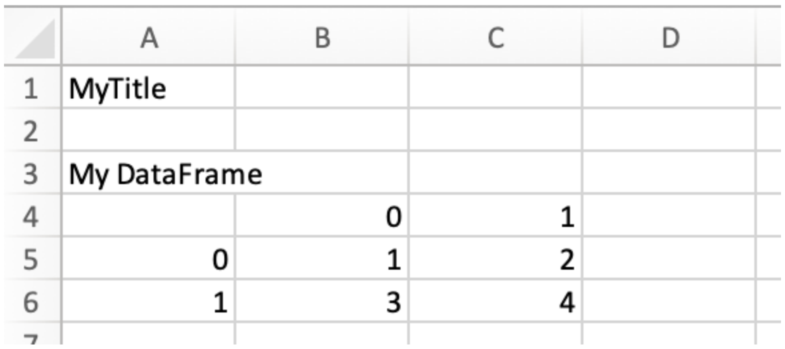


	A	B	C	D
1	{{ title }}			
2				
3	My DataFrame			
4	{{ df }}			
5				

Now run the Python script:

```
python my_template.py
```

This will copy the template and create the following output by replacing the variables in double curly braces with the value from the Python variable:



	A	B	C	D
1	MyTitle			
2				
3	My DataFrame			
4		0	1	
5	0	1	2	
6	1	3	4	

The last line (`wb.to_pdf()`) will print the workbook as PDF, for more details on the options, see *`Book.to_pdf()`*.

Apart from Strings and Pandas DataFrames, you can also use numbers, lists, simple dicts, NumPy arrays, Matplotlib figures and PIL Image objects that have a filename.

By default, xlwings Reports overwrites existing values in templates if there is not enough free space for your variable. If you want your rows to dynamically shift according to the height of your array, use *Frames*.

See also the `API reference`.

13.2 Frames

Frames are vertical containers in which content is being aligned according to their height. That is, within Frames:

- Variables do not overwrite existing cell values as they do without Frames.
- Formatting is applied dynamically, depending on the number of rows your object uses in Excel

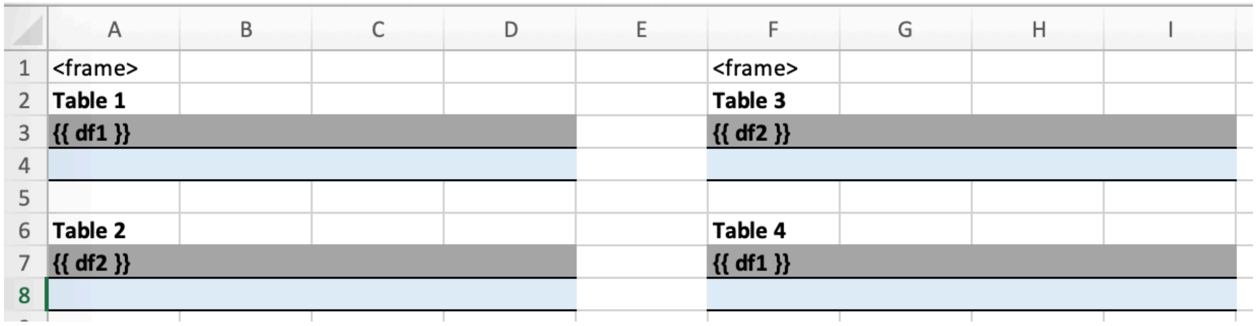
To use Frames, insert `<frame>` into **row 1** of your Excel template wherever you want a new dyanmic column to start. Row 1 will be removed automatically when creating the report. Frames go from one `<frame>` to the next `<frame>` or the right border of the used range.

How Frames behave is best demonstrated with an example: The following screenshot defines two frames. The first one goes from column A to column E and the second one goes from column F to column I, since this is the last column that is used.

You can define and format table-like objects by formatting exactly

- one header and
- one data row

as shown in the screenshot:



	A	B	C	D	E	F	G	H	I
1	<frame>					<frame>			
2	**Table 1**					**Table 3**			
3	**{{ df1 }}**					**{{ df2 }}**			
4									
5									
6	**Table 2**					**Table 4**			
7	**{{ df2 }}**					**{{ df1 }}**			
8									

However, also make sure to check out how to use Excel Tables below, as they make the formatting easier.

Running the following code:

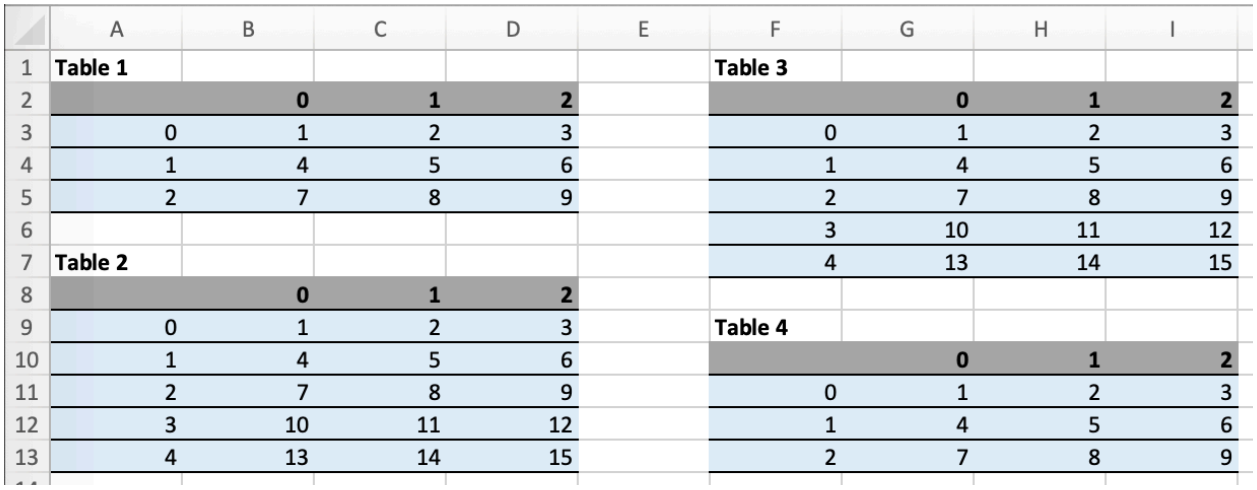
```python
from xlwings.pro.reports import create_report
import pandas as pd

df1 = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
df2 = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15]])
```

( )

( )

```
data = dict(df1=df1, df2=df2)

create_report('my_template.xlsx',
              'my_report.xlsx',
              **data)
```

will generate this report:



	A	B	C	D	E	F	G	H	I
1	**Table 1**					**Table 3**			
2		**0**	**1**	**2**			**0**	**1**	**2**
3	0	1	2	3		0	1	2	3
4	1	4	5	6		1	4	5	6
5	2	7	8	9		2	7	8	9
6						3	10	11	12
7	**Table 2**					4	13	14	15
8		**0**	**1**	**2**					
9	0	1	2	3		**Table 4**			
10	1	4	5	6			**0**	**1**	**2**
11	2	7	8	9		0	1	2	3
12	3	10	11	12		1	4	5	6
13	4	13	14	15		2	7	8	9

13.3 Excel Tables

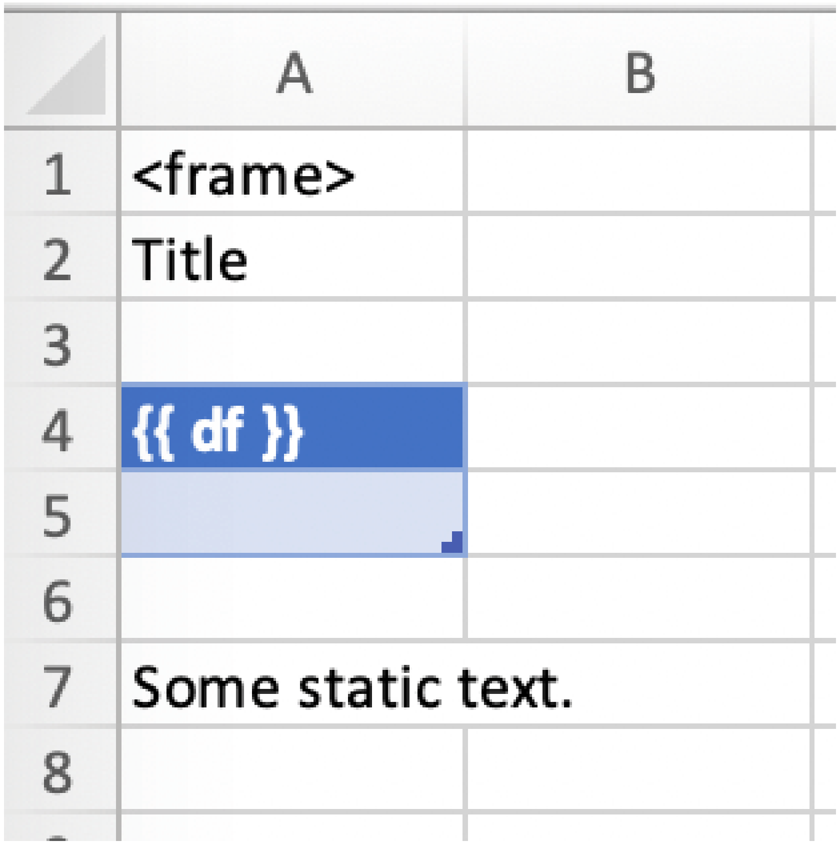
Using Excel tables is the recommended way to format tables as the styling can be applied dynamically across columns and rows. You can also use themes and apply alternating colors to rows/columns. On top of that, they are the easiest way to make the source of a chart dynamic. Go to `Insert` > `Table` and make sure that you activate `My table has headers` before clicking on `OK`. Add the placeholder as usual on the top-left of your Excel table:

Running the following script:

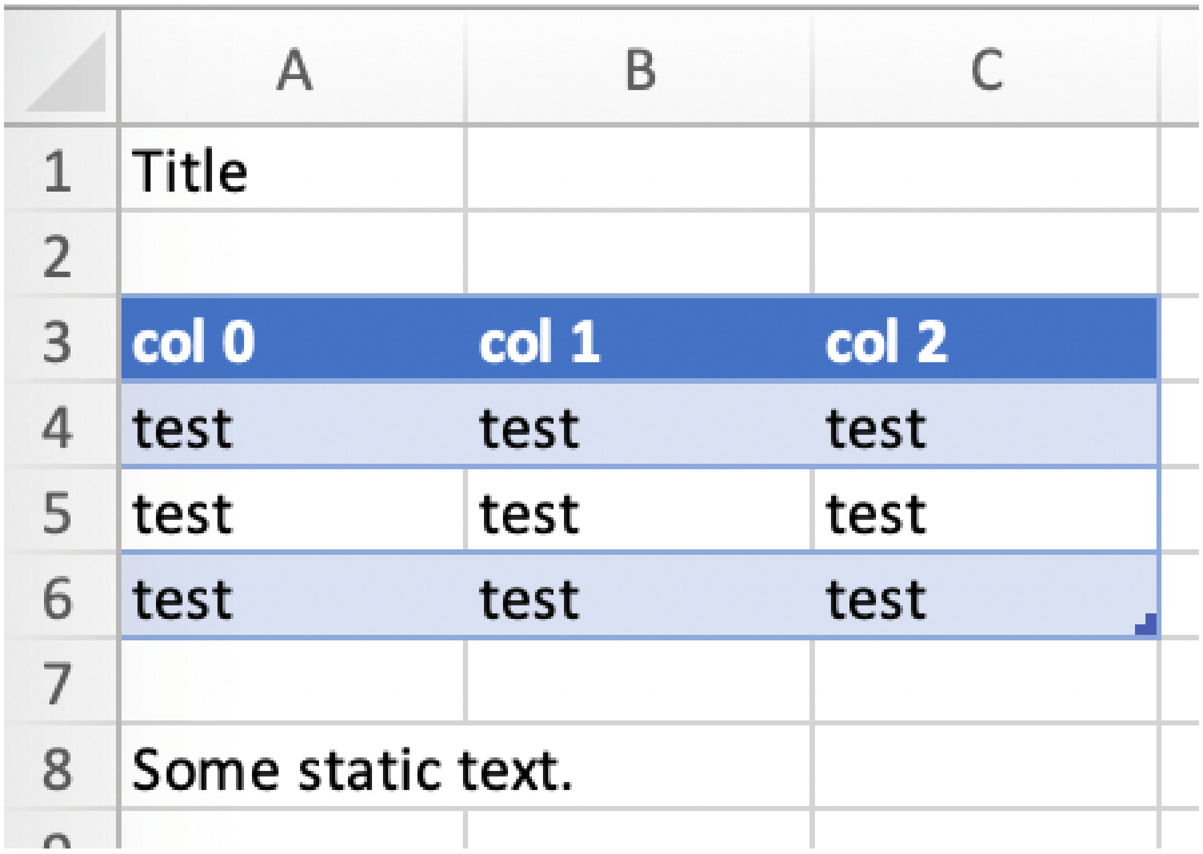
```
from xlwings.pro.reports import create_report
import pandas as pd

nrows, ncols = 3, 3
df = pd.DataFrame(data=nrows * [ncols * ['test']],
                  columns=['col ' + str(i) for i in range(ncols)])

create_report('template.xlsx', 'output.xlsx', df=df.set_index('col 0'))
```

Will produce the following report:

:



	A	B
1	<frame>	
2	Title	
3		
4	{{ df }}	
5		
6		
7	Some static text.	
8		



	A	B	C
1	Title		
2			
3	col 0	col 1	col 2
4	test	test	test
5	test	test	test
6	test	test	test
7			
8	Some static text.		

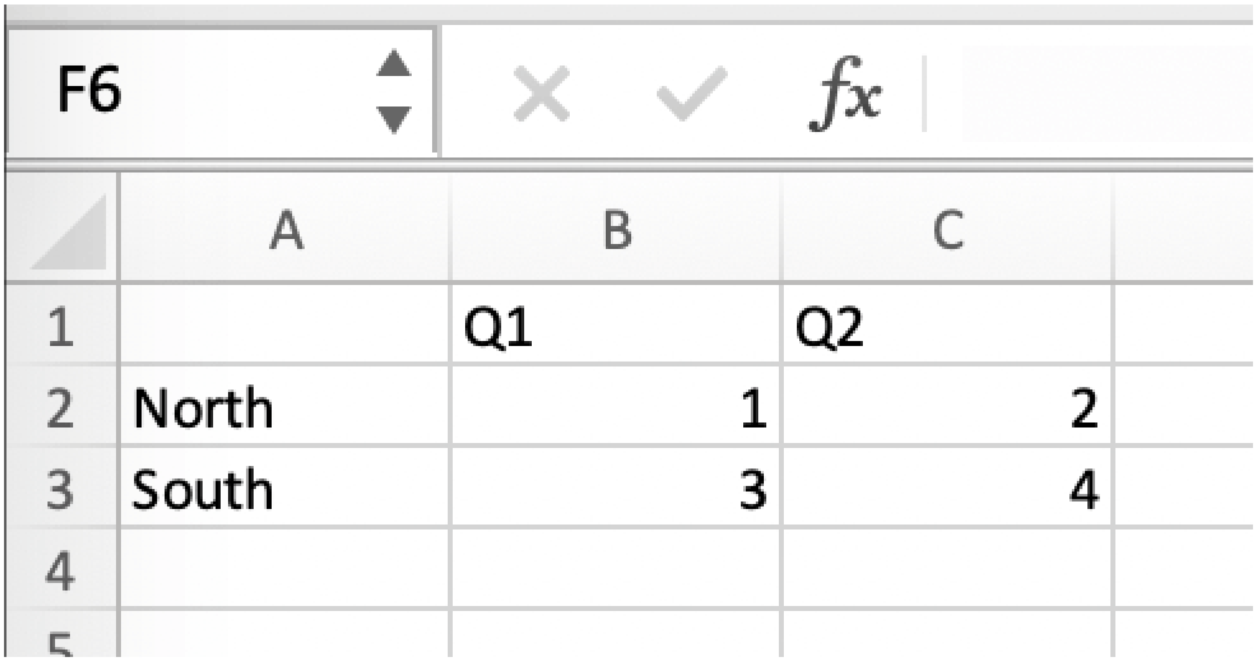
- If you would like to exclude the DataFrame index, make sure to set the index to the first column e.g.: `df.set_index('column_name')`.
- At the moment, you can only assign pandas DataFrames to tables.
- For Excel table support, you need at least version 0.21.0 and the index behavior was changed in 0.21.3

13.4 Excel Charts

Note: To use charts with a dynamic source, you'll need at least xlwings version 0.22.1

To use Excel charts in your reports, follow this process:

1. Add some sample/dummy data to your Excel template:

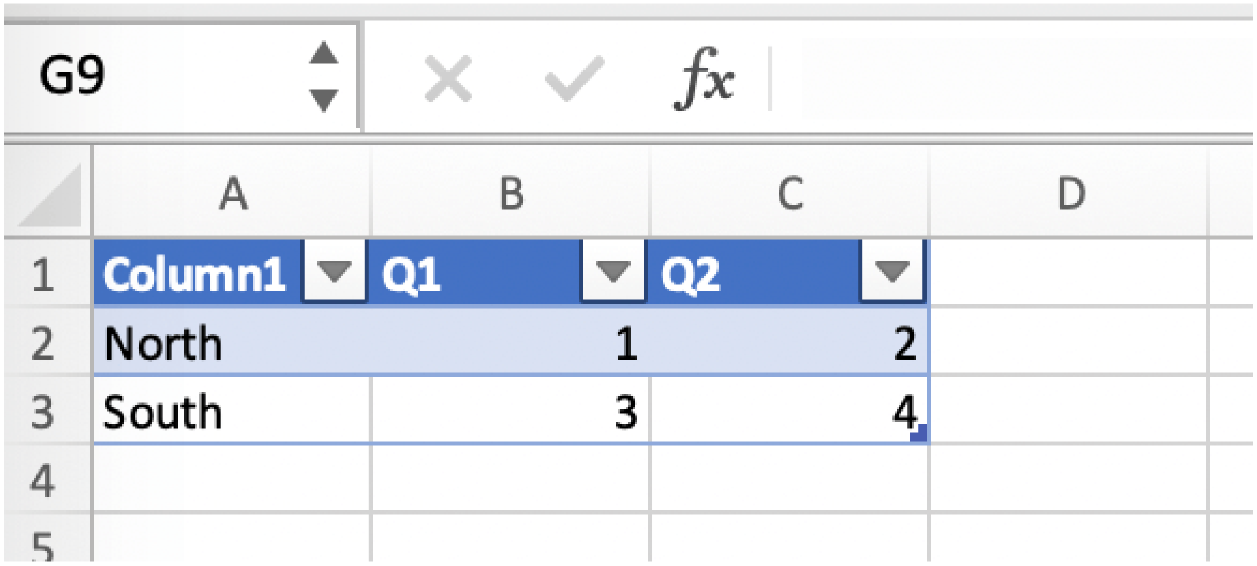


	A	B	C
1		Q1	Q2
2	North	1	2
3	South	3	4
4			

2. If your data source is dynamic, turn it into an Excel Table (`Insert` > `Table`). Make sure you do this *before* adding the chart in the next step.

3. Add your chart and style it:

4. Reduce the Excel table to a 2 x 2 range and add the placeholder in the top-left corner (in our example `chart_data`) . You can leave in some dummy data or clear the values of the Excel table:



	A	B	C	D
1	Column1	Q1	Q2	
2	North	1	2	
3	South	3	4	
4				
5				

5. Assuming your file is called `mytemplate.xlsx` and your sheet `template` like on the previous screenshot, you can run the following code:

```
import xlwings as xw
import pandas as pd

df = pd.DataFrame(data={'Q1': [1000, 2000, 3000],
                        'Q2': [4000, 5000, 6000],
                        'Q3': [7000, 8000, 9000]},
                  index=['North', 'South', 'West'])

wb = xw.Book("mytemplate.xlsx")
sheet = wb.sheets['template'].copy(name='report')
sheet.render_template(chart_data=df)
```

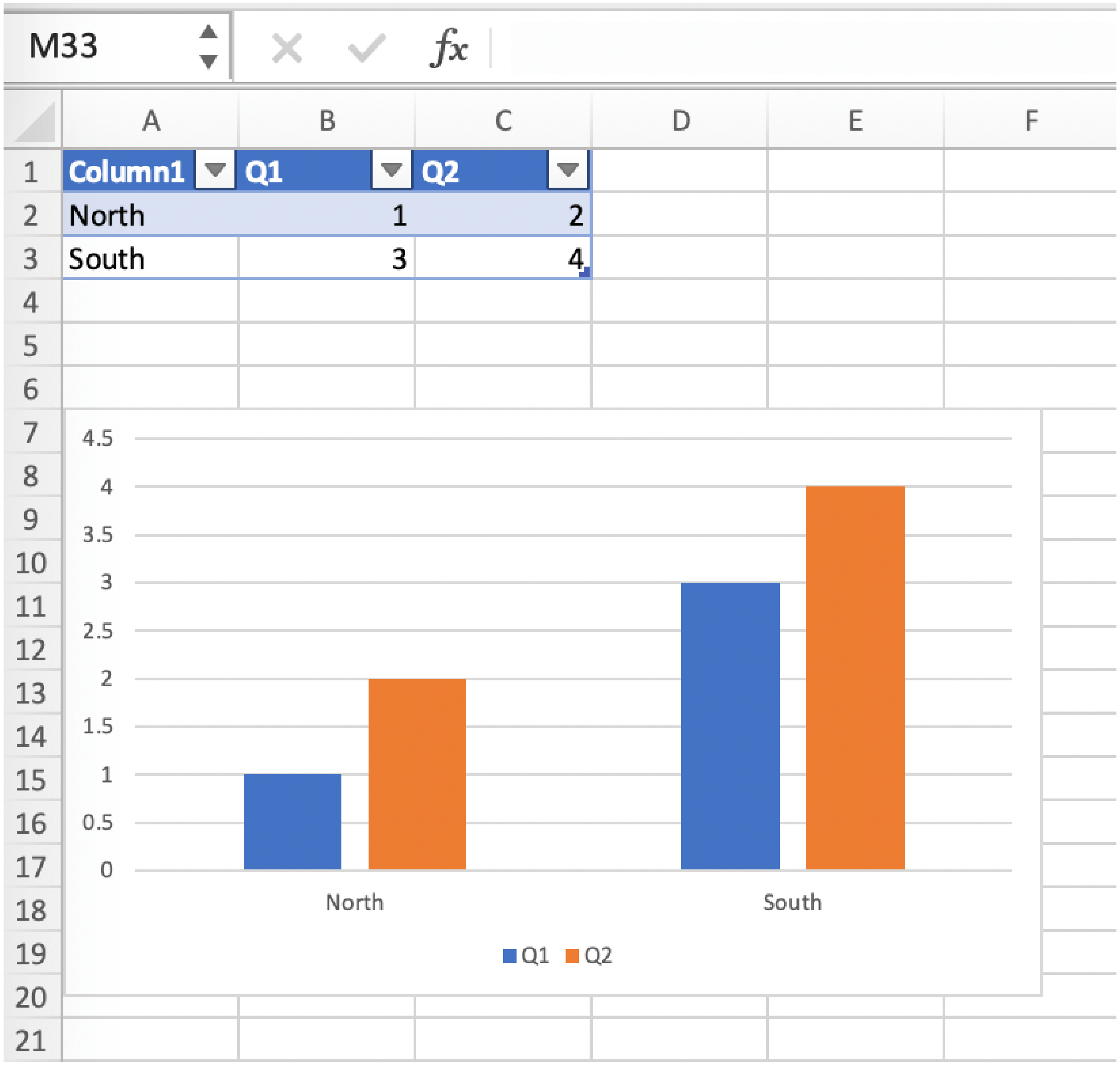
This will produce the following report, with the chart source correctly adjusted:

Note: If you don't want the source data on your report, you might want to place it on a separate sheet. It's easiest if you add and design the chart on the separate sheet, before cutting the chart and pasting it on your report template.

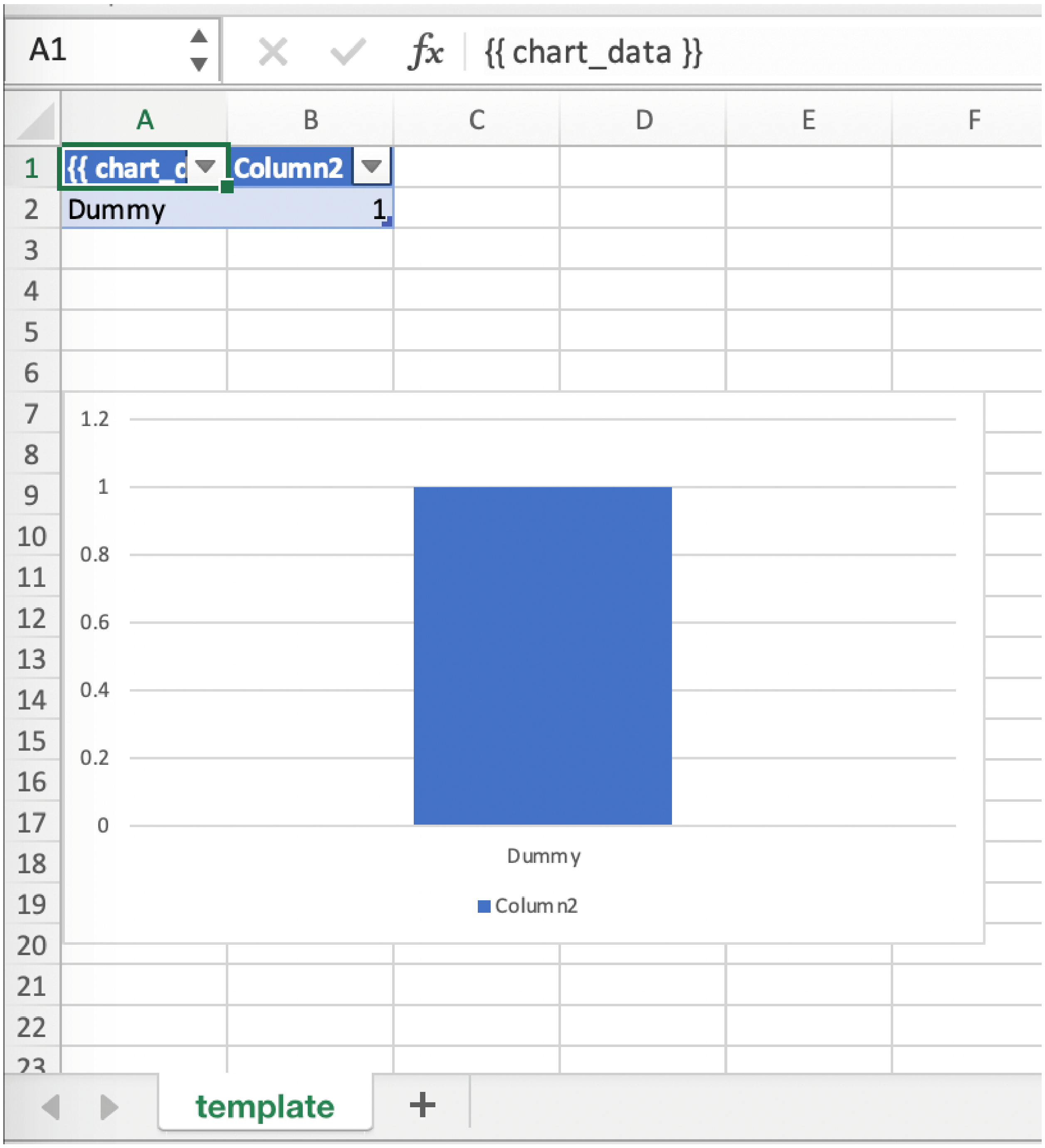
13.5 Shape Text

0.21.4   .

You can also use Shapes like Text Boxes or Rectangles with template text:

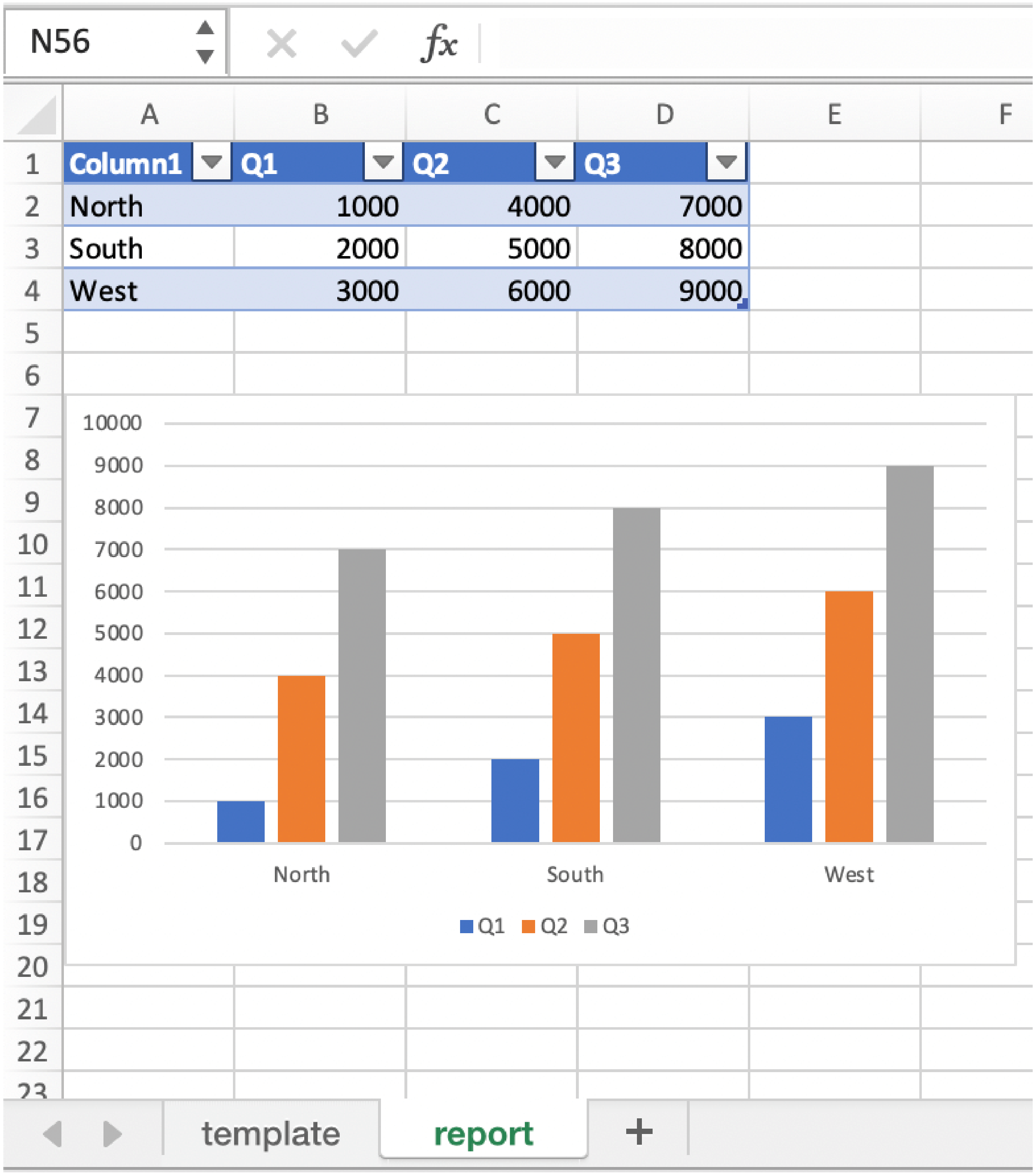


Column1	Q1	Q2
North	1	2
South	3	4



A1 {{ chart_data }}

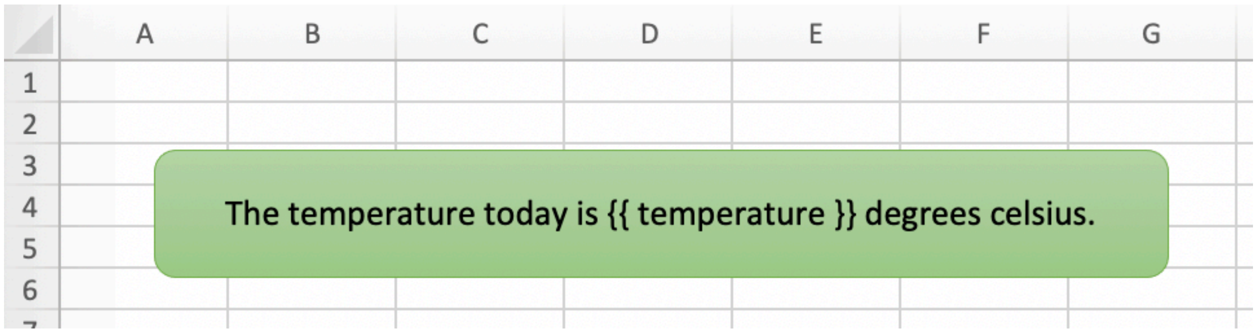
	A	B	C	D	E	F
1	{{ chart_d	Column2				
2	Dummy	1				



Column1	Q1	Q2	Q3
North	1000	4000	7000
South	2000	5000	8000
West	3000	6000	9000

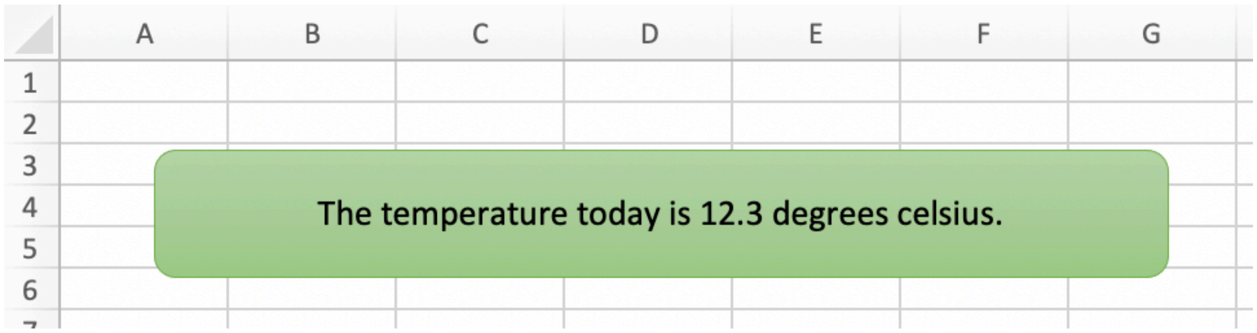
```python
from xlwings.pro.reports import create_report

create_report('template.xlsx', 'output.xlsx', temperature=12.3)
```

This code turns this template:



The temperature today is {{ temperature }} degrees celsius.

into this report:



The temperature today is 12.3 degrees celsius.

While this works for simple text, you will loose the formatting if you have any. To prevent that, use a `Markdown` object, see below.

13.6 Markdown

0.23.0 .

You can format text in cells or shapes via Markdown syntax:

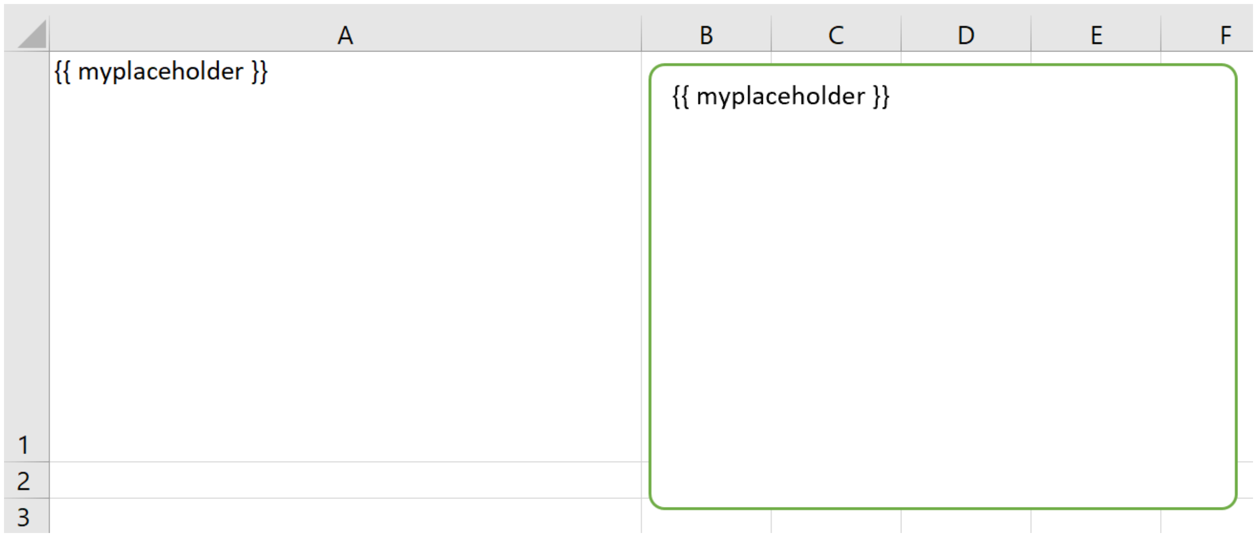
```python
from xlwings.pro import Markdown, MarkdownStyle

mytext = """\
# Title

Text **bold** and *italic*

* A first bullet
* A second bullet
```

( )

( )

```
# Another Title

This paragraph has a line break.
Another line.
"""

# The first sheet requires a shape as shown on the screenshot
sheet = xw.Book("MyTemplate.xlsx").sheets[0]
sheet.render_template(myplaceholder=Markdown(mytext, style)
```

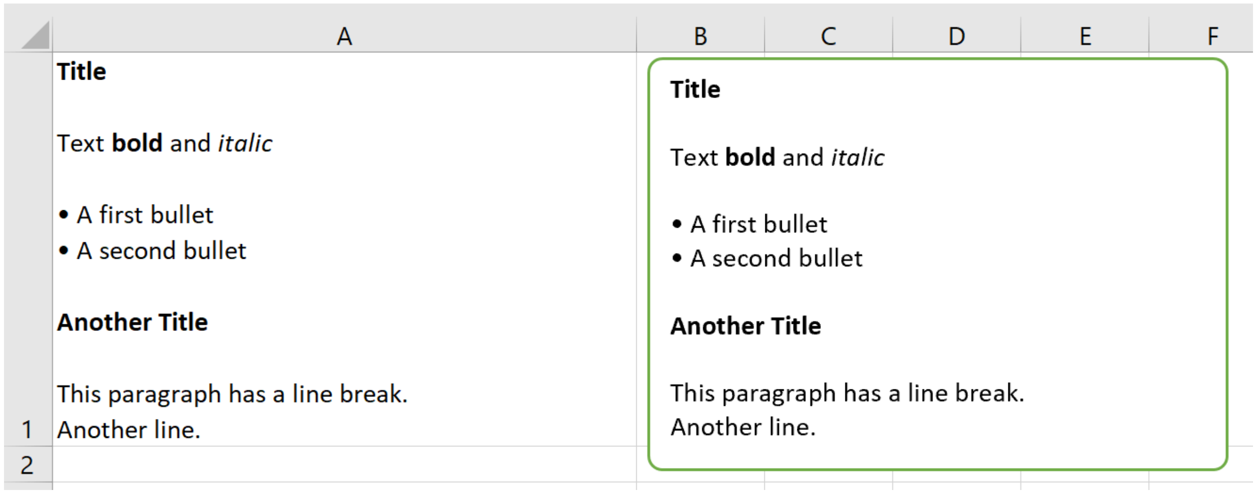
This will render this template with the placeholder in a cell and a shape:



{{ myplaceholder }}

{{ myplaceholder }}

Like this (this uses the default formatting):



Title

Text **bold** and *italic*

- A first bullet
- A second bullet

Another Title

This paragraph has a line break.
Another line.

Title

Text **bold** and *italic*

- A first bullet
- A second bullet

Another Title

This paragraph has a line break.
Another line.

For more on Markdown, especially how to change the styling, see *Markdown Formatting*.

CHAPTER 14

Markdown Formatting

This feature requires xlwings *PRO*.

0.23.0 .

Markdown offers an easy and intuitive way of styling text components in your cells and shapes. For an introduction to Markdown, see e.g., Mastering Markdown.

Markdown support is in an early stage and currently only supports:

- First-level headings
- Bold (i.e., strong)
- Italic (i.e., emphasis)
- Unordered lists

It doesn't support nested objects yet such as 2nd-level headings, bold/italic within bullet points or nested bullet points.

Let's go through an example to see how everything works!

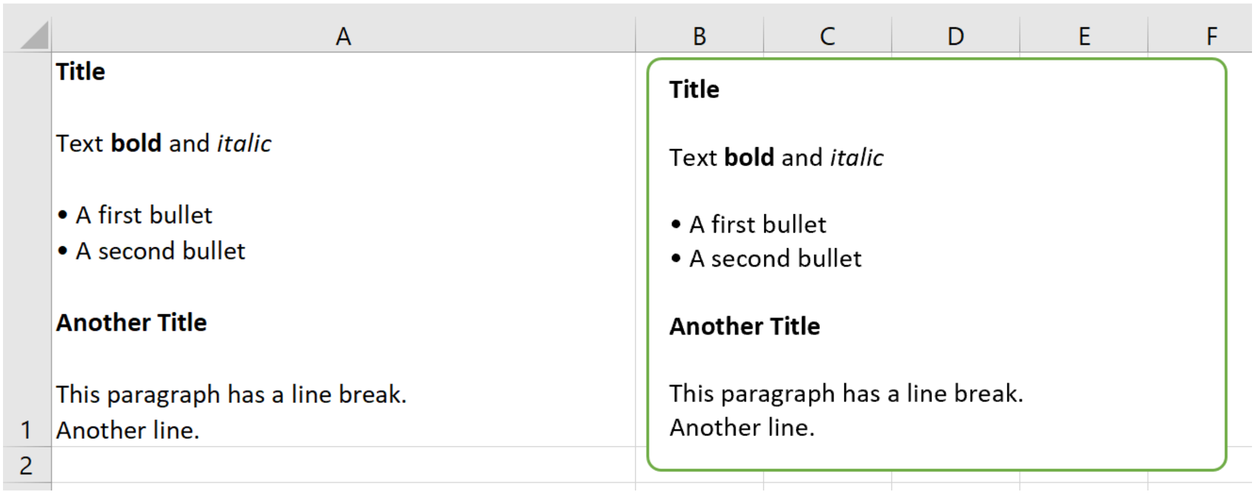
```
from xlwings.pro import Markdown, MarkdownStyle

mytext = """\
# Title

Text **bold** and *italic*

* A first bullet
* A second bullet

```

( )

( )

```
# Another Title

This paragraph has a line break.
Another line.
"""

sheet = xw.Book("Book1.xlsx").sheets[0]

# Range
sheet['A1'].clear()
sheet['A1'].value = Markdown(mytext)

# Shape: The following expects a shape like a Rectangle on the sheet
sheet.shapes[0].text = ""
sheet.shapes[0].text = Markdown(mytext)
```

Running this code will give you this nicely formatted text:



Title

Text **bold** and *italic*

- A first bullet
- A second bullet

Another Title

This paragraph has a line break.
Another line.

Title

Text **bold** and *italic*

- A first bullet
- A second bullet

Another Title

This paragraph has a line break.
Another line.

But why not make things a tad more stylish? By providing a `MarkdownStyle` object, you can define your style. Let's change the previous example like this:

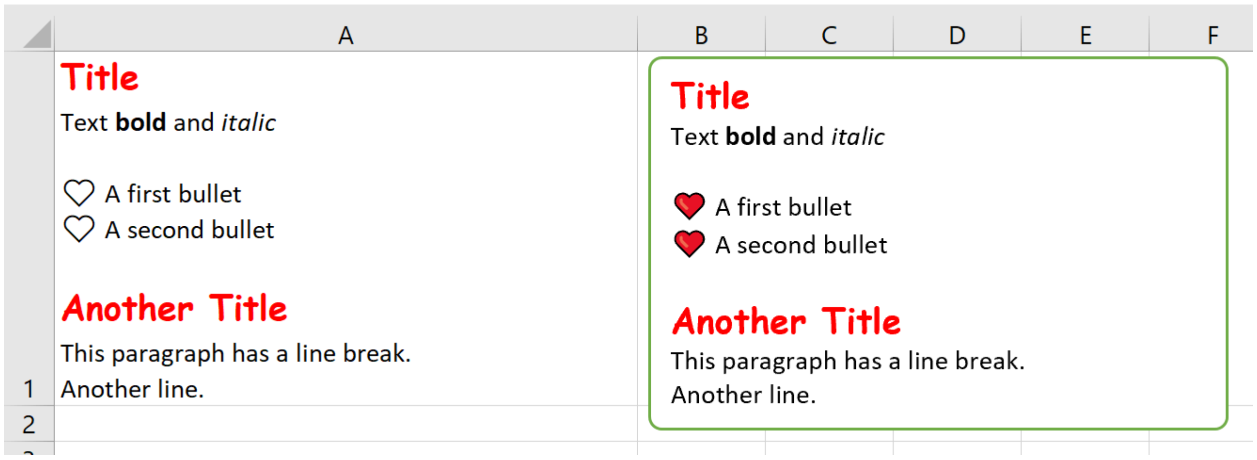
```
from xlwings.pro import Markdown, MarkdownStyle

mytext = """\
# Title

Text **bold** and *italic*

* A first bullet
* A second bullet

# Another Title
```

( )

( )

```
This paragraph has a line break.
Another line.
"""

sheet = xw.Book("Book1.xlsx").sheets[0]

# Styling
style = MarkdownStyle()
style.h1.font.color = (255, 0, 0)
style.h1.font.size = 14
style.h1.font.name = 'Comic Sans MS'  # No, that's not a font recommendation...
style.h1.blank_lines_after = 0
style.unordered_list.bullet_character = '\N{heavy black heart}'  # Emojis are␣
↪fun!

# Range
sheet['A1'].clear()
sheet['A1'].value = Markdown(mytext, style)  # <= provide your style object here

# Shape: The following expects a shape like a Rectangle on the sheet
sheet.shapes[0].text = ""
sheet.shapes[0].text = Markdown(mytext, style)
```

Here is the output of this:



You can override all properties, i.e., you can change the emphasis from italic to a red font or anything else you want:

```
>>> style.strong.bold = False
>>> style.strong.color = (255, 0, 0)
>>> style.strong
strong.color: (255, 0, 0)
```

Markdown objects can also be used with template-based reporting, see *xlwings Reports*.

: macOS currently doesn't support the formatting (bold, italic, color etc.) of Markdown text due to a bug with AppleScript/Excel. The text will be rendered correctly though, including bullet points.

See also the API reference:

- `Markdown class`
- `MarkdownStyle class`

CHAPTER 15

15.1 Zip

0.15.2 .

Python zip UDF zip

zip Excel (`.zip`) Excel xlwings (python)

`PYTHONPATH` :

```
PYTHONPATH, "C:\path\to\myproject.zip"
```

15.2 RunFrozenPython

0.15.2 .

PyInstaller cx_Freeze py2exe Python Python

:

- UDF
- Windows Mac
- V0.15.6 0.15.2

:

```
Sub MySample()
    RunFrozenPython "C:\path\to\dist\myproject\myproject.exe", "arg1 arg2"
End Sub
```

15.3 Embedded Code

This feature requires xlwings *PRO*.

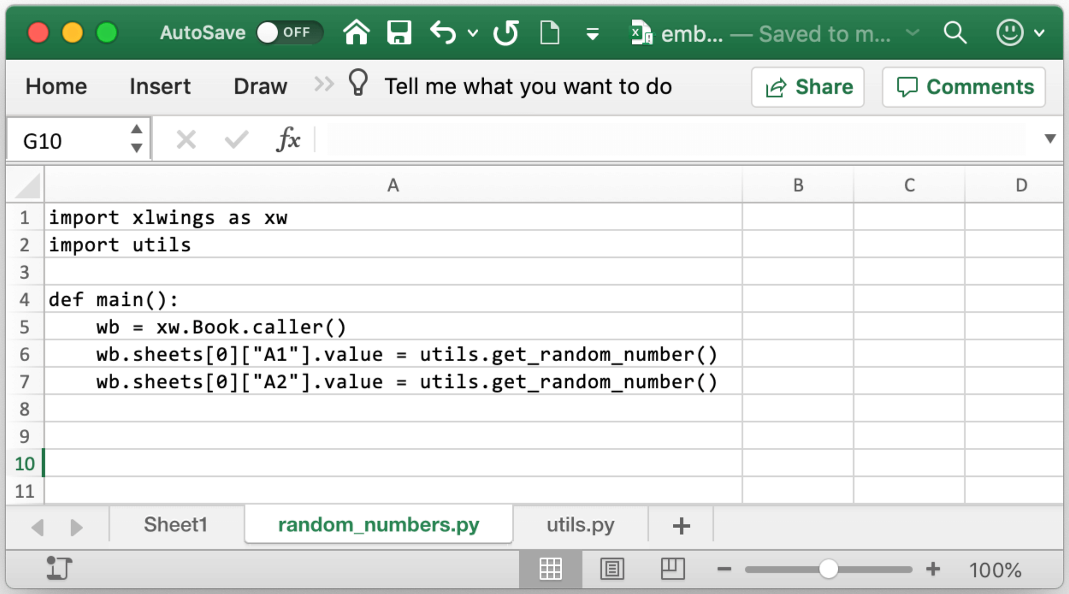
xlwings PRO allows you to store your Python code directly in Excel so you don't have to distribute separate Python files.

On a command line, run the following command which will import all Python files from the current directory and paste them into sheets with the same name of the currently active workbook:

```
$ xlwings code embed
```

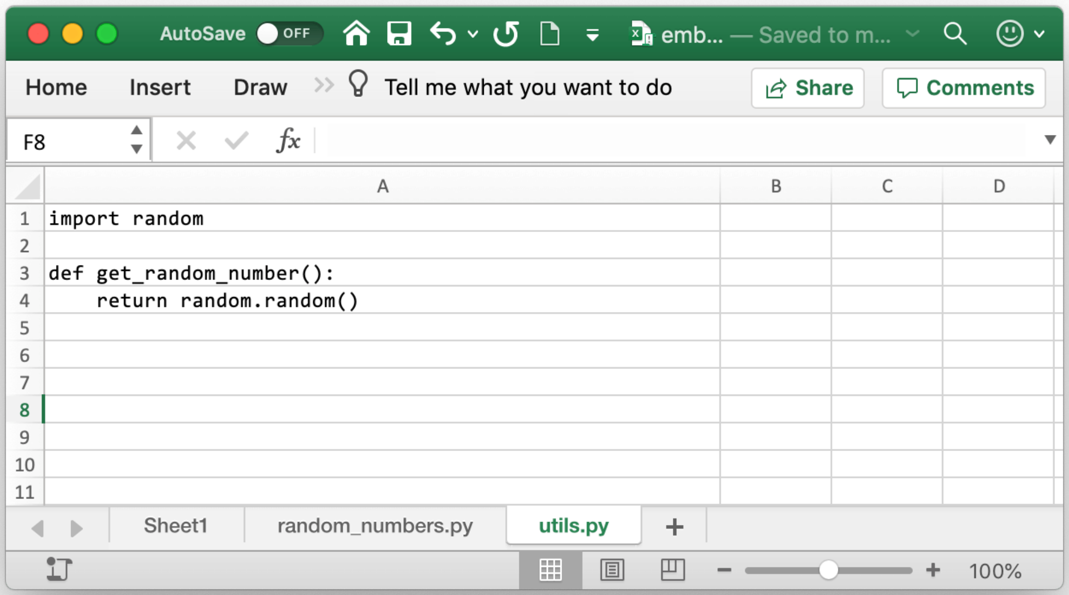
Then, use the VBA function `RunPython ("import mymodule;mymodule.myfunction()")` as usual.

Note that you can have multiple Excel sheets and import them like normal Python files. Consider this example:



```
import xlwings as xw
import utils

def main():
    wb = xw.Book.caller()
    wb.sheets[0]["A1"].value = utils.get_random_number()
    wb.sheets[0]["A2"].value = utils.get_random_number()
```

You can call this function from VBA like so:



```
import random

def get_random_number():
    return random.random()
```

```
Sub RandomNumbers()
    RunPython ("import random_numbers;random_numbers.main()")
End Sub
```

: UDFs modules don't have to be added to the `UDF Modules` explicitly when using embedded code. However, in contrast to how it works with external files, you currently need to re-import the functions when you change them.

: While you can hide your sheets with your code, they will be written to a temporary directory in clear text.

15.4 One-Click Zero-Config Installer

This feature requires xlwings *PRO*.

With xlwings PRO you get access to a private GitHub repository that will build your custom installer in the cloud — no local installation required. Using a custom installer to deploy the Python runtime has the following advantages:

- Zero Python knowledge required from end users

- Zero configuration required by end users
- No admin rights required
- Works for both UDFs and RunPython
- Works for external distribution
- Easy to deploy updates

15.4.1 End User Instructions

- **Installing**

  Give the end user your Excel workbook and the installer. The user only has to double-click the installer and confirm a few prompts — no configuration is required.

- **Updating**

  If you use the embedded code feature (see: *Embedded Code*), you can deploy updates by simply giving the user a new Excel file. Only when you change a dependency, you will need to create a new installer.

- **Uninstalling**

  The application can be uninstalled again via Windows Settings > Apps & Features.

15.4.2 Build the Installer

Before you can build the installer, the project needs to be configured correctly, see below.

In the GitHub repo, go to `x releases` > `Draft/Create a new release`. Add a version like `1.0.0` to `Tag version`, then hit `Publish release`.

Wait a few minutes and refresh the page: the installer will appear under the release from where you can download it. You can follow the progress under the `Actions` tab.

15.4.3 Configuration

Excel file

You can add your Excel file to the repository if you like but it's not a requirement. Configure the Excel file as follows:

- Add the standalone xlwings VBA module, e.g. via `xlwings quickstart project --standalone`
- Make sure that in the VBA editor (`Alt-F11`) under `Tools` > `References` xlwings is unchecked
- Rename the `_xlwings.conf` sheet into `xlwings.conf`
- In the `xlwings.conf` sheet, as `Interpreter`, set the following value: `%LOCALAPPDATA%\project` while replacing `project` with the name of your project

- If you like, you can hide the `xlwings.conf` sheet

Source code

Source code can either be embedded in the Excel file (see *Embedded Code*) or added to the `src` directory. The first option requires `xlwings-pro` in `requirements.txt`, the second option will also work with `xlwings`.

Dependencies

Add your dependencies to `requirements.txt`. For example:

```
xlwings==0.18.0
numpy==1.18.2
```

Code signing (optional)

Using a code sign certificate will show a verified publisher in the installation prompt. Without it, it will show an unverified publisher.

- Store your code sign certificate as `sign_cert_file` in the root of this repository (make sure your repo is private).
- Go to `Settings` > `Secrets` and add the password as `code_sign_password`.

Project details

Update the following under `.github/main.yml`:

```
PROJECT:
APP_PUBLISHER:
```

Python version

Set your Python version under `.github/main.yml`:

```
python-version: '3.7'
architecture: 'x64'
```

15.5 Deployment Key

This feature requires xlwings *PRO*.

If you have an xlwings PRO developer license, you can generate a deployment key. A deployment key allows you to send an xlwings PRO tool to an end user without them requiring a paid license. A deployment key is also perpetual, i.e. doesn't expire like a developer license.

In return, a deployment key only works with the version of xlwings that was used to generate the deployment key. A developer can generate new deployment keys for new versions of xlwings as long as they have an active xlwings PRO subscription.

: You need a paid developer license to generate a deployment key. A trial license won't work.

To create a deployment key, run the following command:

```
xlwings license deploy
```

Then paste the generated key into the xlwings config as `LICENSE_KEY`. For deployment purposes, usually the best place to do that is on a sheet called `xlwings.conf`, but you can also use an `xlwings.conf` file in either the same folder or in the `.xlwings` folder within the user's home folder. To use an environment variable, use `XLWINGS_LICENSE_KEY`. See also *User Settings*.

CHAPTER 16

16.1 : dll

:

1) `xlwings32-<version>.dll` `xlwings64-<version>.dll` `python.exe` `pip` `conda` , .
2) `Interpreter` `python` Python `'python' is not recognized as an internal or external command, operable program or batch file.`, `python.exe` windows (https://www.computerhope.com/issues/ch000549.htm) / (`C:\Users\MyUser\anaconda\pythonw.exe`)

16.2 Issue: Couldn't find the local location of your OneDrive

:

On either the `xlwings.conf` sheet or on the `xlwings.conf` file under your home folder (for location see *User Config: Ribbon/Config File*), add the following setting:

```
"ONEDRIVE", "C:\path\to\OneDrive"
```

Note: Don't use quotes on the `xlwings.conf` sheet.

CHAPTER 17

xlwings PRO

The purpose of xlwings PRO is to finance the continued maintenance and enhancement of xlwings. This will allow you to rely on the package without being left with the dreaded "this library currently has no active maintainers" message that happens to too many open-source packages after a couple of years.

xlwings PRO offers access to additional functionality. All PRO features are marked with xlwings *PRO* in the docs.

: To get access to the additional functionality of xlwings PRO, you need a license key and at least xlwings v0.19.0. Everything under the `xlwings.pro` subpackage is distributed under a commercial license.

17.1 PRO Features

- *`Table.update()`*: An easy way to keep an Excel table in sync with a pandas DataFrame
- *Embedded Code*: Store your Python source code directly in Excel for easy deployment.
- *xlwings Reports*: A template based reporting mechanism, allows business users to change the layout of the report without having to change Python code.
- *Plotly static charts*: Support for Plotly static charts.
- *One-Click Zero-Config Installer*: Guarantees that the end user does not need to know anything about Python.

17.2 More Infos

- Pricing: https://www.xlwings.org/pricing
- Trial license key: https://www.xlwings.org/trial

CHAPTER 18

v0.7.0 Excel **xlwings.Range** **User Defined Functions (UDFs)**

Converters are explicitly set in the `options` method when manipulating `Range` objects or in the `@xw.arg` and `@xw.ret` decorators when using UDFs. If no converter is specified, the default converter is applied when reading. When writing, xlwings will automatically apply the correct converter (if available) according to the object's type that is being written to Excel. If no converter is found for that type, it falls back to the default converter.

```
>>> import xlwings as xw
```

:

	xw.Range	**UDFs**
	`xw.Range.options(convert=None, **kwargs).value`	`@arg('x', convert=None, **kwargs)`
writing	`xw.Range.options(convert=None, **kwargs).value = myvalue`	`@ret(convert=None, **kwargs)`

: (kwargs) `numbers` `DataFrame` `index` :

```
xw.Range('A1:C3').options(pd.DataFrame, index=False, numbers=int).value
```

18.1

:

- floats unicode datetime None
- / : [None, 1.0, 'a string']
- 2 [[None, 1.0, 'a string'], [None, 2.0, 'another string']]

- **ndim**

  1 2

```
>>> import xlwings as xw
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = [[1, 2], [3, 4]]
>>> sht.range('A1').value
1.0
>>> sht.range('A1').options(ndim=1).value
[1.0]
>>> sht.range('A1').options(ndim=2).value
[[1.0]]
>>> sht.range('A1:A2').value
[1.0 3.0]
>>> sht.range('A1:A2').options(ndim=2).value
[[1.0], [3.0]]
```

- **numbers**

  float int

```
>>> sht.range('A1').value = 1
>>> sht.range('A1').value
1.0
>>> sht.range('A1').options(numbers=int).value
1
```

float

UDF :

```
@xw.func
@xw.arg('x', numbers=int)
def myfunction(x):
    # all numbers in x arrive as int
    return x
```

Note: Excel *int* 5 5 4 Python *int* *raw* *int*

- **dates**

  `datetime.datetime` `datetime.date`

  – Range:

  ```
  >>> import datetime as dt
  >>> sht.range('A1').options(dates=dt.date).value
  ```

  – UDFs: `@xw.arg('x', dates=dt.date)`

  `datetime.datetime`

```
>>> my_date_handler = lambda year, month, day, **kwargs: "%04i-%02i-%02i" %␣
↪(year, month, day)
>>> sht.range('A1').options(dates=my_date_handler).value
'2017-02-20'
```

- **empty**

  `None`

  – Range: `>>> sht.range('A1').options(empty='NA').value`

  – UDFs: `@xw.arg('x', empty='NA')`

- **transpose**

  Excel

  – Range: `sht.range('A1').options(transpose=True).value = [1, 2, 3]`

  – UDFs:

  ```
  @xw.arg('x', transpose=True)
  @xw.ret(transpose=True)
  def myfunction(x):
      # x will be returned unchanged as transposed both when reading and␣
  ↪writing
      return x
  ```

- **expand**

  `table` , `vertical` `horizontal`

```
>>> import xlwings as xw
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = [[1,2], [3,4]]
>>> rng1 = sht.range('A1').expand()
>>> rng2 = sht.range('A1').options(expand='table')
>>> rng1.value
[[1.0, 2.0], [3.0, 4.0]]
>>> rng2.value
```

( )

( )

```
[[1.0, 2.0], [3.0, 4.0]]
>>> sht.range('A3').value = [5, 6]
>>> rng1.value
[[1.0, 2.0], [3.0, 4.0]]
>>> rng2.value
[[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]
```

: `expand` `Range` UDF

18.2

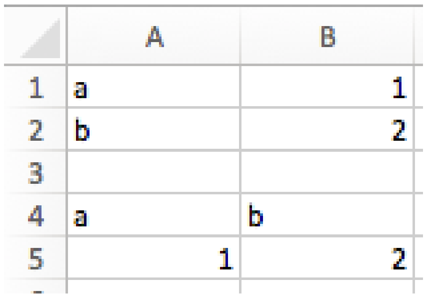
xlwings **dictionaries** ( ), **NumPy arrays**(NumPy), **Pandas Series**(Pandas) **DataFrames** (ndim)

It is also possible to write and register a custom converter for additional types, see below.

`xlwings.Range` UDF

18.2.1

Excel `transpose`



	A	B
1	a	1
2	b	2
3		
4	a	b
5	1	2

```
>>> sht = xw.sheets.active
>>> sht.range('A1:B2').options(dict).value
{'a': 1.0, 'b': 2.0}
>>> sht.range('A4:B5').options(dict, transpose=True).value
{'a': 1.0, 'b': 2.0}
```

`dict` `collections` `OrderedDict`

18.2.2 Numpy

options: `dtype=None, copy=True, order=None, ndim=None`

3 np.array() ndim ( ) 1 2

:

```
>>> import numpy as np
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').options(transpose=True).value = np.array([1, 2, 3])
>>> sht.range('A1:A3').options(np.array, ndim=2).value
array([[ 1.],
       [ 2.],
       [ 3.]])
```

18.2.3 Pandas

options: dtype=None, copy=False, index=1, header=True

2 pd.Series() Pandas ndim

index:

Excel

True False

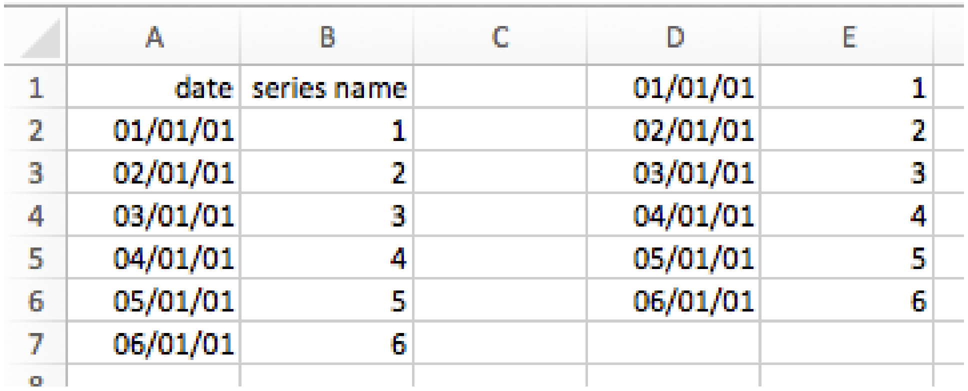
header:

Excel False

True False

index header , 1 True

:



	A	B	C	D	E
1	date	series name		01/01/01	1
2	01/01/01	1		02/01/01	2
3	02/01/01	2		03/01/01	3
4	03/01/01	3		04/01/01	4
5	04/01/01	4		05/01/01	5
6	05/01/01	5		06/01/01	6
7	06/01/01	6			

```
>>> sht = xw.Book().sheets[0]
>>> s = sht.range('A1').options(pd.Series, expand='table').value
>>> s
date
2001-01-01    1
2001-01-02    2
```

( )

( )

```
2001-01-03    3
2001-01-04    4
2001-01-05    5
2001-01-06    6
Name: series name, dtype: float64
>>> sht.range('D1', header=False).value = s
```

18.2.4 Pandas DataFrame

options: dtype=None, copy=False, index=1, header=1

2 pd.DataFrame() ndim Pandas DataFrame ndim=2

index:

Excel

True False

header:

Excel

True False

index header , 1 True

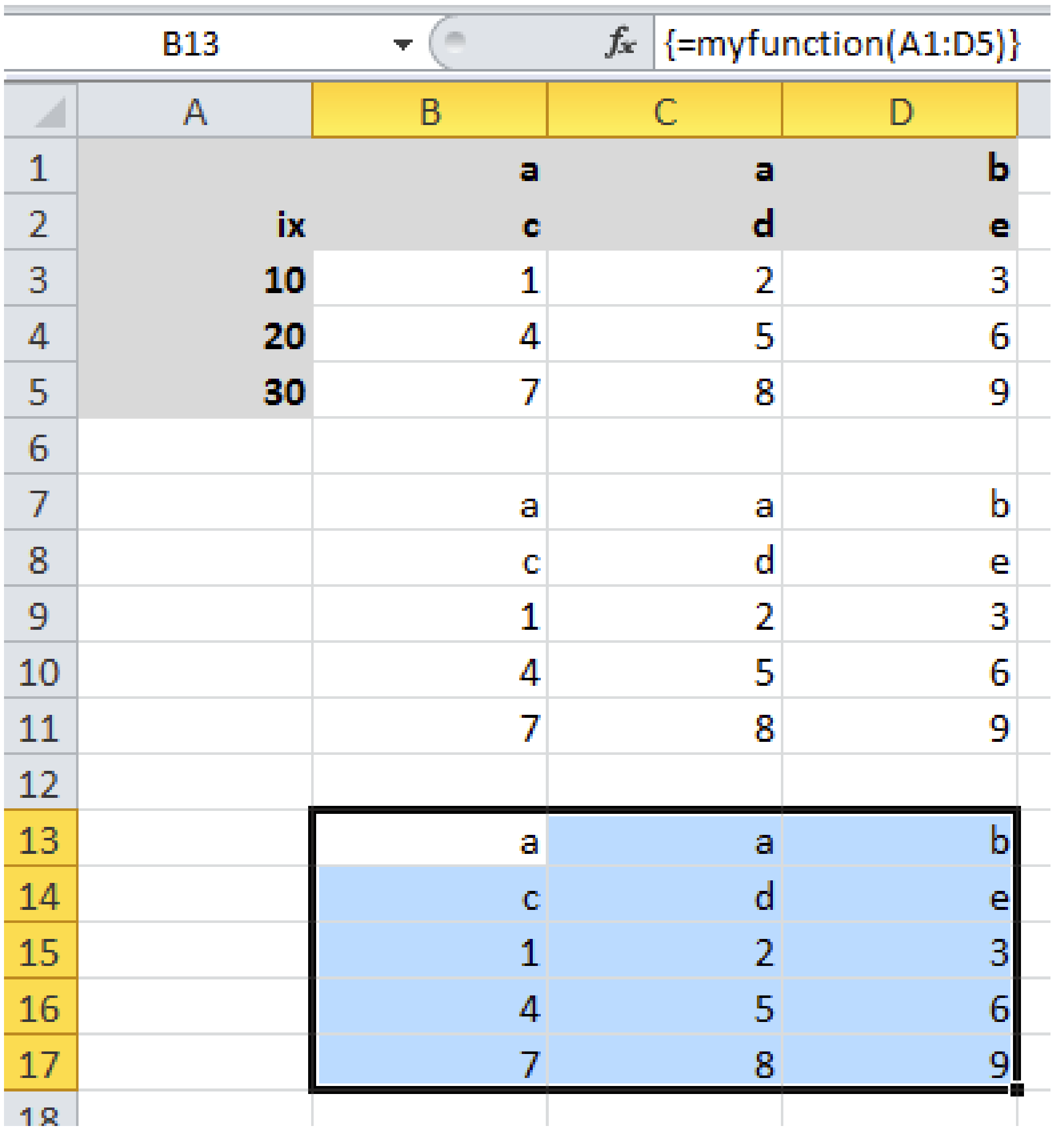
:

```
>>> sht = xw.Book().sheets[0]
>>> df = sht.range('A1:D5').options(pd.DataFrame, header=2).value
>>> df
    a     b
    c  d  e
ix
10  1  2  3
20  4  5  6
30  7  8  9

# Writing back using the defaults:
>>> sht.range('A1').value = df

# Writing back and changing some of the options, e.g. getting rid of the index:
>>> sht.range('B7').options(index=False).value = df
```

UDF (Range('A13'))

```
@xw.func
@xw.arg('x', pd.DataFrame, header=2)
@xw.ret(index=False)
```

( )


B13 {=myfunction(A1:D5)}

	A	B	C	D
1		a	a	b
2	ix	c	d	e
3	10	1	2	3
4	20	4	5	6
5	30	7	8	9
6				
7		a	a	b
8		c	d	e
9		1	2	3
10		4	5	6
11		7	8	9
12				
13		a	a	b
14		c	d	e
15		1	2	3
16		4	5	6
17		7	8	9
18				

( )

```
def myfunction(x):
    # x is a DataFrame, do something with it
    return x
```

18.2.5 xw.Range ‘ ’

” ”

- xlwings.Range :

  ```
  @xw.func
  @xw.arg('x', 'range')
  def myfunction(x):
      return x.formula
  ```

  xlwings.Range x

- (Windows pywin32 Mac appscript) :

  ```
  >>> sht.range('A1:B2').value
  [[1.0, 'text'], [datetime.datetime(2016, 2, 1, 0, 0), None]]

  >>> sht.range('A1:B2').options('raw').value  # or sht.range('A1:B2').raw_
  ↪value
  ((1.0, 'text'), (pywintypes.datetime(2016, 2, 1, 0, 0, tzinfo=TimeZoneInfo(
  ↪'GMT Standard Time', True)), None))
  ```

18.3

- xlwings.conversion.Converter
- read_value write_value
  - read_value value (Base converter) base
  - write_value value Excel base

options xw.Range.options (xw.Range('A1').options(myoption='some value')) UDF @arg @ret :

```
from xlwings.conversion import Converter

class MyConverter(Converter):

    @staticmethod
```

( )

( )

```
    def read_value(value, options):
        myoption = options.get('myoption', default_value)
        return_value = value  # Implement your conversion here
        return return_value

    @staticmethod
    def write_value(value, options):
        myoption = options.get('myoption', default_value)
        return_value = value  # Implement your conversion here
        return return_value
```

- base (base) : DictCoverter, NumpyArrayConverter, PandasDataFrameConverter, PandasSeriesConverter
- **(a)** / **(b)**

  DataFrame :

```
from xlwings.conversion import Converter, PandasDataFrameConverter

class DataFrameDropna(Converter):

    base = PandasDataFrameConverter

    @staticmethod
    def read_value(builtin_df, options):
        dropna = options.get('dropna', False)  # set default to False
        if dropna:
            converted_df = builtin_df.dropna()
        else:
            converted_df = builtin_df
        # This will arrive in Python when using the DataFrameDropna converter␣
↪for reading
        return converted_df

    @staticmethod
    def write_value(df, options):
        dropna = options.get('dropna', False)
        if dropna:
            converted_df = df.dropna()
        else:
            converted_df = df
        # This will be passed to the built-in PandasDataFrameConverter when␣
↪writing
        return converted_df
```

:

```
# Fire up a Workbook and create a sample DataFrame
sht = xw.Book().sheets[0]
df = pd.DataFrame([[1.,10.],[2.,np.nan], [3., 30.]])
```

- DataFrames :

```
# Write
sht.range('A1').value = df

# Read
sht.range('A1:C4').options(pd.DataFrame).value
```

- DataFrameDropna :

```
# Write
sht.range('A7').options(DataFrameDropna, dropna=True).value = df

# Read
sht.range('A1:C4').options(DataFrameDropna, dropna=True).value
```

- ( ):

```
DataFrameDropna.register('df_dropna')

# Write
sht.range('A12').options('df_dropna', dropna=True).value = df

# Read
sht.range('A1:C4').options('df_dropna', dropna=True).value
```

- DataFrameDropna DataFrames ( ):

```
DataFrameDropna.register(pd.DataFrame)

# Write
sht.range('A13').options(dropna=True).value = df

# Read
sht.range('A1:C4').options(pd.DataFrame, dropna=True).value
```

UDF :

```
@xw.func
@arg('x', DataFrameDropna, dropna=True)
@ret(DataFrameDropna, dropna=True)
def myfunction(x):
    # ...
    return x
```

: Python Excel Excel/COM Python

Pipelines are internally defined by `Accessor` classes. A Converter is just a special Accessor which converts to/from a particular type by adding an extra stage to the pipeline of the default Accessor. For example, the `PandasDataFrameConverter` defines how a list of lists (as delivered by the default Accessor) should be turned into a Pandas DataFrame.

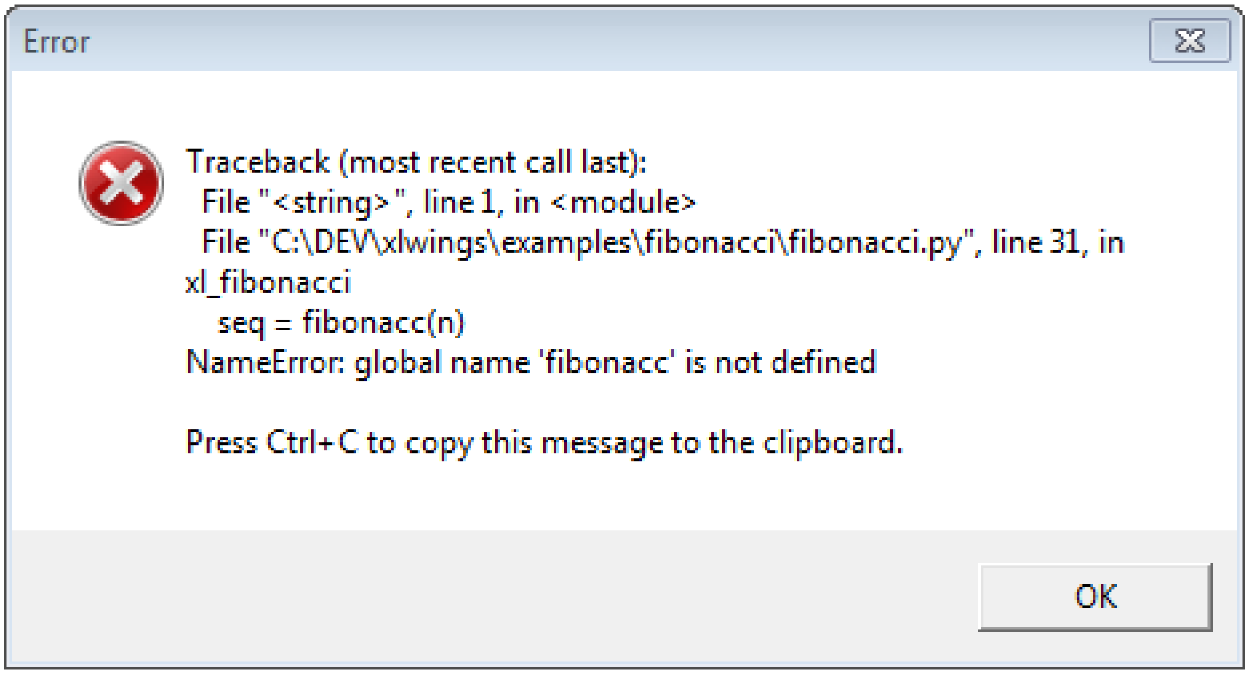
`Converter` ( ) `Accessor`

CHAPTER 19

xlwings Python

- **RunPython**: `RunPython` Python `mock_caller` Excel Python
- **UDFs**: xlwings

Excel Python



Error

```
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "C:\DEV\xlwings\examples\fibonacci\fibonacci.py", line 31, in
xl_fibonacci
    seq = fibonacc(n)
NameError: global name 'fibonacc' is not defined

Press Ctrl+C to copy this message to the clipboard.
```

: Mac `xlwings` (`/Users/<User>/Library/Containers/`

com.microsoft.Excel/Data/xlwings.log)

19.1 RunPython

Python my_module.py

```
# my_module.py
import os
import xlwings as xw

def my_macro():
    wb = xw.Book.caller()
    wb.sheets[0].range('A1').value = 1

if __name__ == '__main__':
    # Expects the Excel file next to this source file, adjust accordingly.
    xw.Book('myfile.xlsm').set_mock_caller()
    my_macro()
```

my_macro() Python Excel RunPython

```
Sub my_macro()
    RunPython "import my_module; my_module.my_macro()"
End Sub
```

19.2 UDF

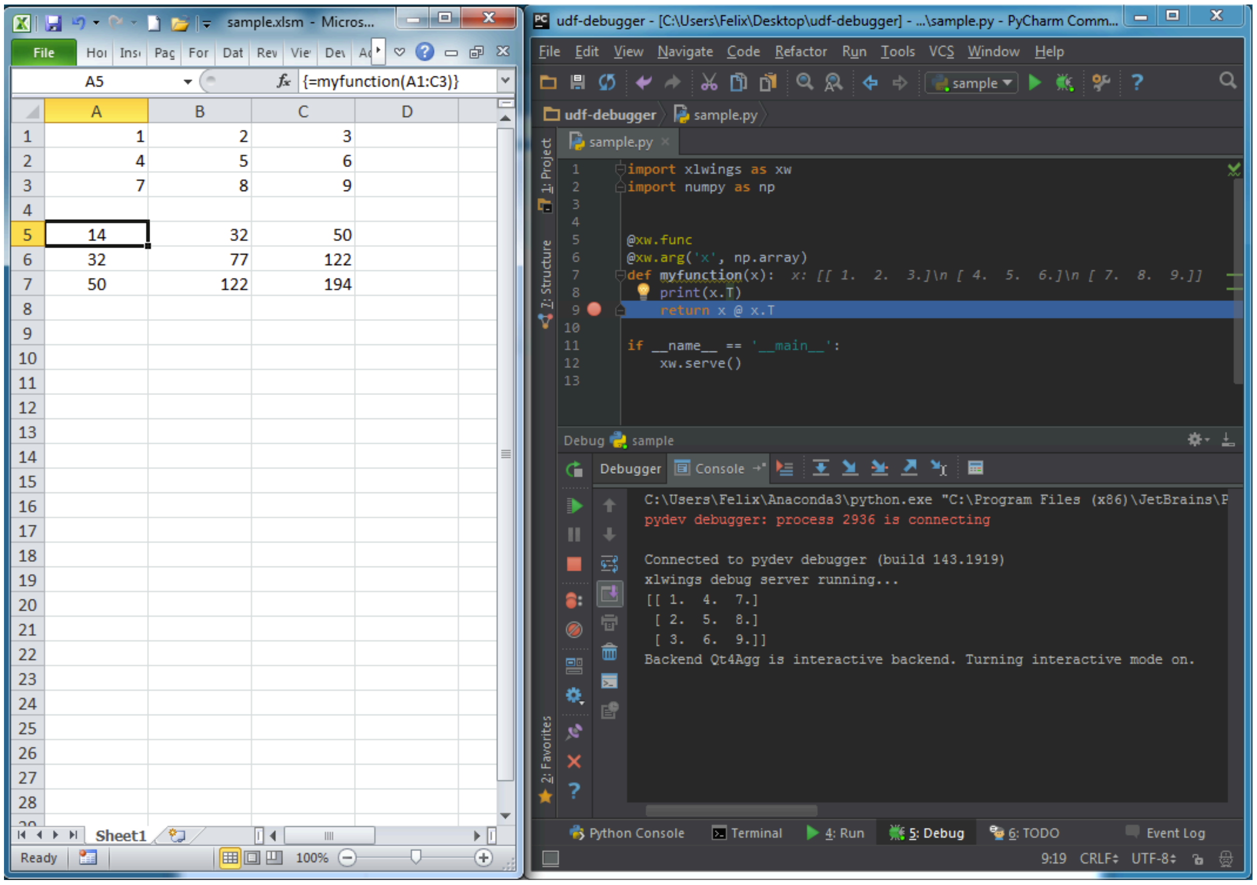
Windows UDF xlwings *Add-in* *&* *Settings* VBA Debug UDFs , Python Python (PyCharm PyDev):

```
if __name__ == '__main__':
    xw.serve()
```

(Ctrl-Alt-F9)

PyCharm

:



A5 {=myfunction(A1:C3)}

	A	B	C	D
1	1	2	3	
2	4	5	6	
3	7	8	9	
4				
5	14	32	50	
6	32	77	122	
7	50	122	194	

udf-debugger - [C:\Users\Felix\Desktop\udf-debugger] - ...\sample.py - PyCharm Comm...

```python
import xlwings as xw
import numpy as np


@xw.func
@xw.arg('x', np.array)
def myfunction(x):  x: [[ 1.  2.  3.]\n [ 4.  5.  6.]\n [ 7.  8.  9.]]
    print(x.T)
    return x @ x.T

if __name__ == '__main__':
    xw.serve()
```

```
C:\Users\Felix\Anaconda3\python.exe "C:\Program Files (x86)\JetBrains\P
pydev debugger: process 2936 is connecting

Connected to pydev debugger (build 143.1919)
xlwings debug server running...
[[ 1.  4.  7.]
 [ 2.  5.  8.]
 [ 3.  6.  9.]]
Backend Qt4Agg is interactive backend. Turning interactive mode on.
```

CHAPTER 20

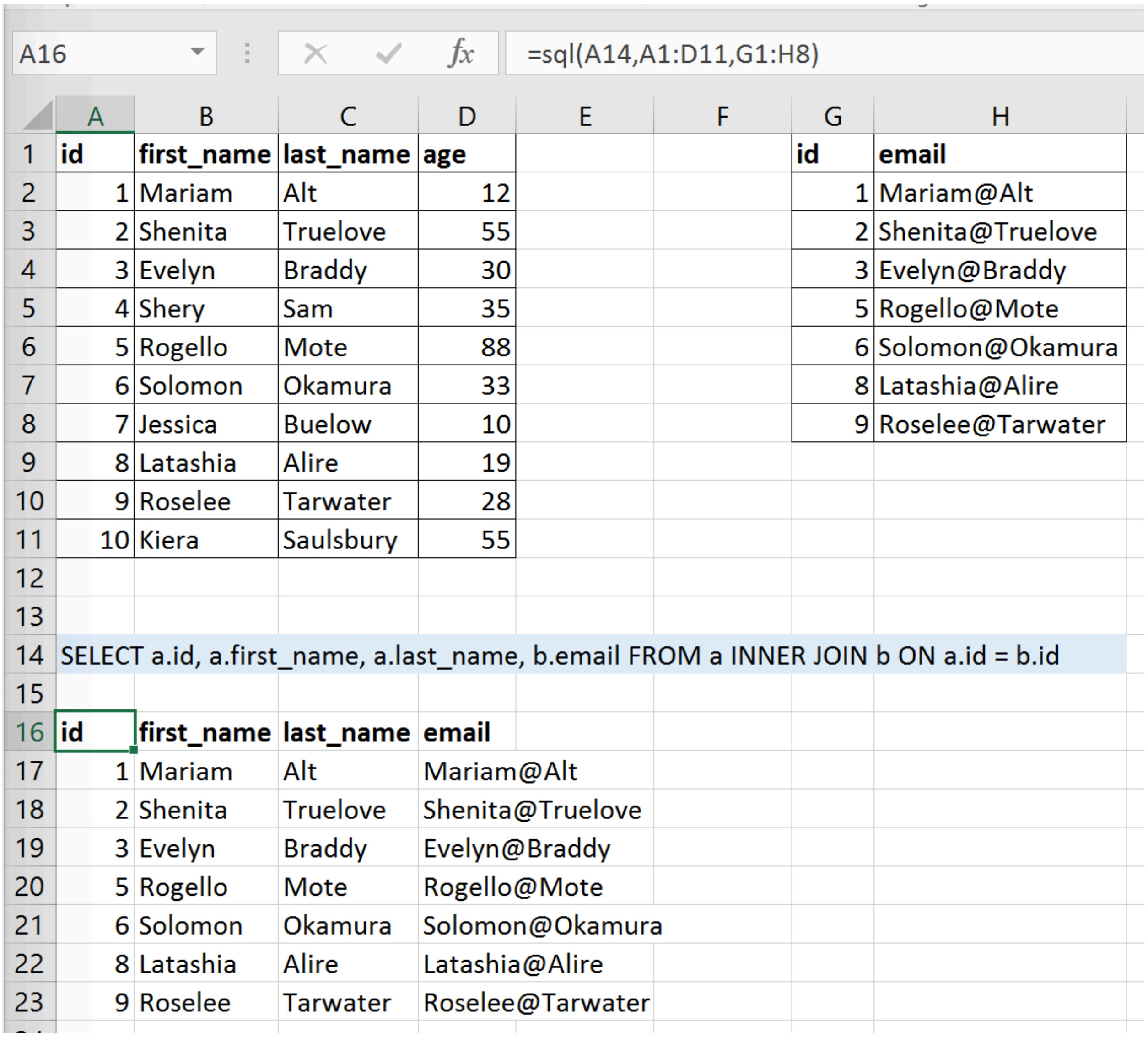
UDF RunPython xlwings xlwings VBA UDF

20.1 In-Excel SQL

xlwings Excel SQL(in-Excel SQL) (sqlite) :

```
=sql(SQL Statement, table a, table b, ...)
```

UDF Windows



A16 =sql(A14,A1:D11,G1:H8)

	A	B	C	D	E	F	G	H
1	id	first_name	last_name	age			id	email
2	1	Mariam	Alt	12			1	Mariam@Alt
3	2	Shenita	Truelove	55			2	Shenita@Truelove
4	3	Evelyn	Braddy	30			3	Evelyn@Braddy
5	4	Shery	Sam	35			5	Rogello@Mote
6	5	Rogello	Mote	88			6	Solomon@Okamura
7	6	Solomon	Okamura	33			8	Latashia@Alire
8	7	Jessica	Buelow	10			9	Roselee@Tarwater
9	8	Latashia	Alire	19				
10	9	Roselee	Tarwater	28				
11	10	Kiera	Saulsbury	55				
12								
13								
14	SELECT a.id, a.first_name, a.last_name, b.email FROM a INNER JOIN b ON a.id = b.id							
15								
16	id	first_name	last_name	email				
17	1	Mariam	Alt	Mariam@Alt				
18	2	Shenita	Truelove	Shenita@Truelove				
19	3	Evelyn	Braddy	Evelyn@Braddy				
20	5	Rogello	Mote	Rogello@Mote				
21	6	Solomon	Okamura	Solomon@Okamura				
22	8	Latashia	Alire	Latashia@Alire				
23	9	Roselee	Tarwater	Roselee@Tarwater				

CHAPTER 21

Custom Add-ins

0.22.0 .

Custom add-ins work on Windows and macOS and are white-labeled xlwings add-ins that include all your `RunPython` functions and UDFs (as usual, UDFs work on Windows only). You can build add-ins with and without an Excel ribbon.

The useful thing about add-in is that UDFs and RunPython calls will be available in all workbooks right out of the box without having to add any references via the VBA editor's `Tools > References...`. You can also work with standard `xlsx` files rather than `xlsm` files. This tutorial assumes you're familiar with how xlwings and its configuration works.

21.1 Quickstart

Start by running the following command on a command line (to create an add-in without a ribbon, you would leave away the `--ribbon` flag):

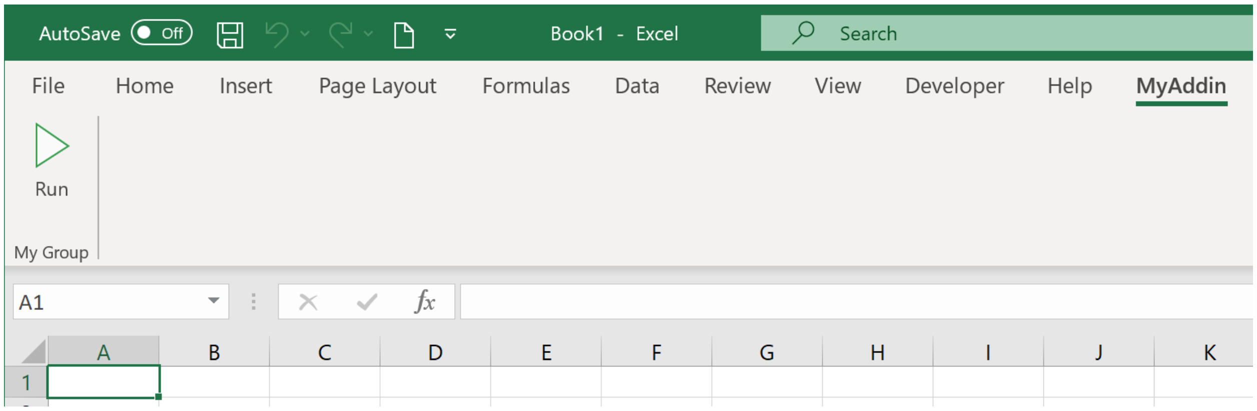
```
$ xlwings quickstart myproject --addin --ribbon
```

This will create the familiar quickstart folder with a Python file and an Excel file, but this time, the Excel file is in the `xlam` format.

- Double-click the Excel add-in to open it in Excel
- Add a new empty workbook (`Ctrl+N` on Windows or `Command+N` on macOS)

You should see a new ribbon tab called `MyAddin` like this:

The add-in and VBA project are currently always called `myaddin`, no matter what name you chose in the quickstart command. We'll see towards the end of this tutorial how we can change that, but for now we'll stick with it.



Compared to the xlwings add-in, the custom add-in offers an additional level of configuration: the configuration sheet of the add-in itself which is the easiest way to configure simple add-ins with a static configuration.

Let's open the VBA editor by clicking on `Alt+F11` (Windows) or `Option+F11` (macOS). In our project, select `ThisWorkbook`, then change the Property `IsAddin` from `True` to `False`, see the following screenshot:

This will make the sheet `_myaddin.conf` visible (again, we'll see how to change the name of `myaddin` at the end of this tutorial):

- Activate the sheet config by renaming it from `_myaddin.conf` to `myaddin.conf`
- Set your `Interpreter_Win/_Mac` or `Conda` settings (you may want to take them over from the xlwings settings for now)

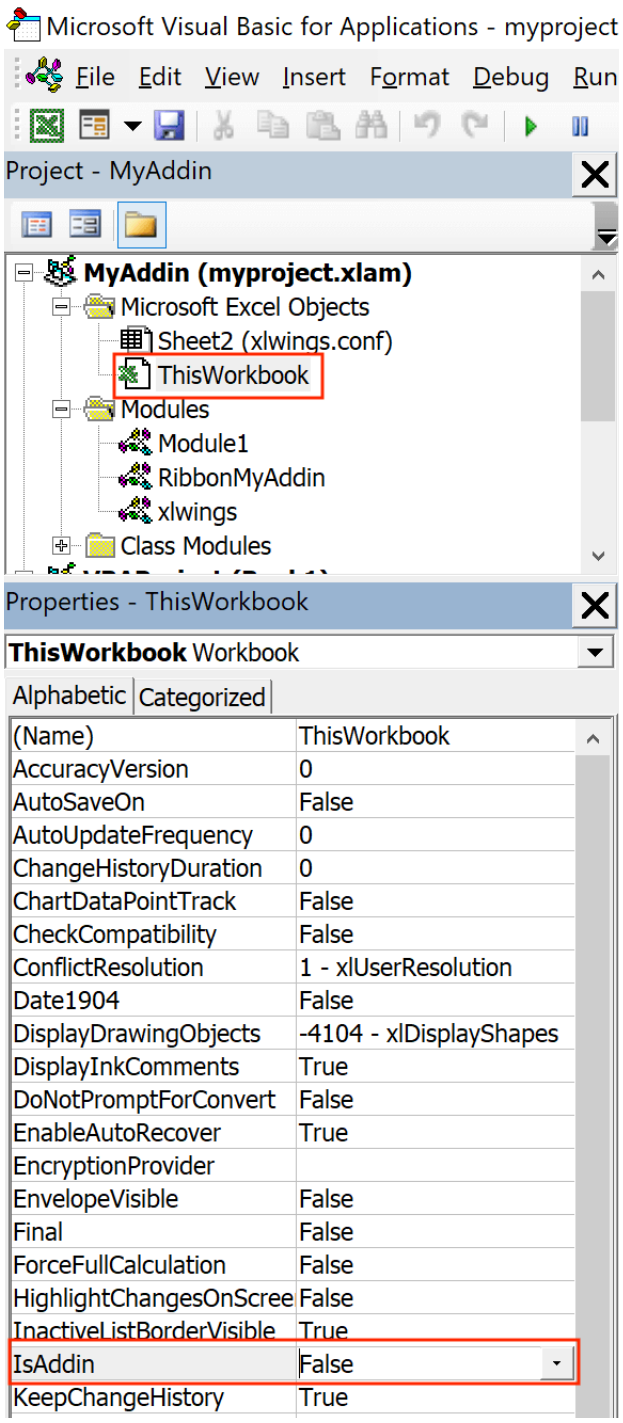
Once done, switch back to the VBA editor, select `ThisWorkbook` again, and change `IsAddin` back to `True` before you save your add-in from the VBA editor. Switch back to Excel and click the `Run` button under the `My Addin` ribbon tab and if you've configured the Python interpreter correctly, it will print `Hello xlwings!` into cell `A1` of the active workbook.

21.2 Changing the Ribbon menu

To change the buttons and items in the ribbon menu or the Backstage View, download and install the Office RibbonX Editor. While it is only available for Windows, the created ribbons will also work on macOS. Open your add-in with it so you can change the XML code that defines your buttons etc. You will find a good tutorial here. The callback function for the demo `Run` button is in the `RibbonMyAddin` VBA module that you'll find in the VBA editor.

21.3 Importing UDFs

To import your UDFs into the custom add-in, run the `ImportPythonUDFsToAddin` Sub towards the end of the `xlwings` module (click into the Sub and hit `F5`). Remember, you only have to do this whenever you change the function name, argument or decorator, so your end users won't have to deal with this.



(Name)	ThisWorkbook
AccuracyVersion	0
AutoSaveOn	False
AutoUpdateFrequency	0
ChangeHistoryDuration	0
ChartDataPointTrack	False
CheckCompatibility	False
ConflictResolution	1 - xlUserResolution
Date1904	False
DisplayDrawingObjects	-4104 - xlDisplayShapes
DisplayInkComments	True
DoNotPromptForConvert	False
EnableAutoRecover	True
EncryptionProvider	
EnvelopeVisible	False
Final	False
ForceFullCalculation	False
HighlightChangesOnScree	False
InactiveListBorderVisible	True
IsAddin	False
KeepChangeHistory	True

If you are only deploying UDFs via your add-in, you probably don't need a Ribbon menu and can leave away the `--ribbon` flag in the `quickstart` command.

21.4 Configuration

As mentioned before, configuration works the same as with xlwings, so you could have your users override the default configuration we did above by adding a `myaddin.conf` sheet on their workbook or you could use the `myaddin.conf` file in the user's home directory. For details see *Add-in & Settings*.

21.5 Installation

If you want to permanently install your add-in, you can do so by using the xlwings CLI:

```
$ xlwings addin install --file C:\path\to\your\myproject.xlam
```

This, however, means that you will need to adjust the `PYTHONPATH` for it to find your Python code (or move your Python code to somewhere where Python looks for it—more about that below under deployment). The command will copy your add-in to the `XLSTART` folder, a special folder from where Excel will open all files everytime you start it.

21.6 Renaming your add-in

Admittedly, this part is a bit cumbersome for now. Let's assume, we would like to rename the addin from `MyAddin` to `Demo`:

- In the `xlwings` VBA module, change `Public Const PROJECT_NAME As String = "myaddin"` to `Public Const PROJECT_NAME As String = "demo"`. You'll find this line at the top, right after the `Declare` statements.
- If you rely on the `myaddin.conf` sheet for your configuration, rename it to `demo.conf`
- Right-click the VBA project, select `MyAddin Properties...` and rename the `Project Name` from `MyAddin` to `Demo`.
- If you use the ribbon, you want to rename the `RibbonMyAddin` VBA module to `RibbonDemo`. To do this, select the module in the VBA editor, then rename it in the `Properties` window. If you don't see the Properties window, hit `F4`.
- Open the add-in in the Office RibbonX Editor (see above) and replace all occurrences of `MyAddin` with `Demo` in the XML code.

And finally, you may want to rename your `myproject.xlam` file in the Windows explorer, but I assume you have already run the quickstart command with the correct name, so this won't be necessary.

21.7 Deployment

By far the easiest way to deploy your add-in to your end-users is to build an installer via the xlwings PRO offering. This will take care of everything and your end users literally just need to double-click the installer and they are all set (no existing Python installation required and no manual installation of the add-in or adjusting of settings required).

If you want it the free (but hard) way, you either need to build an installer yourself or you need your users to install Python and the add-in and take care of placing the Python code in the correct directory. This normally involves tweaking the following settings, for example in the `myaddin.conf` sheet:

- `Interpreter_Win/_Mac`: if your end-users have a working version of Python, you can use environment variables to dynamically resolve to the correct path. For example, if they have Anaconda installed in the default location, you could use the following configuration:

```
Conda Path: %USERPROFILE%\anaconda3
Conda Env: base
Interpreter_Mac: $HOME/opt/anaconda3/bin/python
```

- `PYTHONPATH`: since you can't have your Python source code in the `XLSTART` folder next to the add-in, you'll need to adjust the `PYTHONPATH` setting and add the folder to where the Python code will be. You could point this to a shared drive or again make use of environment variables so the users can place the file into a folder called `MyAddin` in their home directory, for example. However, you can also place your Python code where Python looks for it, for example by placing them in the `site-packages` directory of the Python distribution—an easy way to achieve this is to build a Python package that you can install via `pip`.

CHAPTER 22

0.13.0 .

22.1

xlwings v0.13.0 xlwings macOS

:

```python
import threading
from queue import Queue
import xlwings as xw

num_threads = 4


def write_to_workbook():
    while True:
        rng = q.get()
        rng.value = rng.address
        print(rng.address)
        q.task_done()


q = Queue()

for i in range(num_threads):
    t = threading.Thread(target=write_to_workbook)
```

( )

( )

```
    t.daemon = True
    t.start()

for cell in ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'A10']:
    # THIS DOESN'T WORK - passing xlwings objects to threads will fail!
    rng = xw.Book('Book1.xlsx').sheets[0].range(cell)
    q.put(rng)

q.join()
```

Book :

```
import threading
from queue import Queue
import xlwings as xw

num_threads = 4


def write_to_workbook():
    while True:
        cell_ = q.get()
        xw.Book('Book1.xlsx').sheets[0].range(cell_).value = cell_
        print(cell_)
        q.task_done()


q = Queue()

for i in range(num_threads):
    t = threading.Thread(target=write_to_workbook)
    t.daemon = True
    t.start()

for cell in ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'A10']:
    q.put(cell)

q.join()
```

22.2

: Multiprocessing is only supported on Windows!

:

```
from multiprocessing import Pool
import xlwings as xw


def write_to_workbook(cell):
    xw.Book('Book1.xlsx').sheets[0].range(cell).value = cell
    print(cell)


if __name__ == '__main__':
    with Pool(4) as p:
        p.map(write_to_workbook,
              ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'A10'])
```

CHAPTER 23

xlwings :

1) Most importantly, open an issue on GitHub. Adding functionality should be user driven, so only if you tell us about what you're missing, it's eventually going to find its way into the library. By the way, we also appreciate pull requests!

2) Workaround: in essence, xlwings is just a smart wrapper around pywin32 on Windows and appscript on Mac. You can access the underlying objects by calling the `api` property:

```
>>> sheet = xw.Book().sheets[0]
>>> sheet.api
<COMObject <unknown>>  # Windows/pywin32
app(pid=2319).workbooks['Workbook1'].worksheets[1]  # Mac/appscript
```

This works accordingly for the other objects like `sheet.range('A1').api` etc.

VBA pywin32 (VBA) appscript (VBA) (!)
2) 1) Github xlwings (Python)

23.1 VBA `Range.WrapText`

```
# Windows
sheet.range('A1').api.WrapText = True

# Mac
sheet.range('A1').api.wrap_text.set(True)
```

CHAPTER 24

xlwings Office

Excel Office (Outlook Access) VBA xlwings Python

:	v0.12.0	Windows UDF RunPython

24.1

1) Python Excel (*User Defined Functions (UDFs)*)
2) `Alt-F11` VBA VBA `xlwings_udfs` `Export File( )...` `xlwings_udfs.bas`
3) Office Access `Alt-F11` VBA VBA Project `Import File( )...,` `Microsoft Excel` `Microsoft Access` `Microsoft Outlook` : `#Const App = "Microsoft Access"`
4) xlwings VBA (`xlwings.bas`) xlwings :

```
>>> import xlwings as xw
>>> xlwings.__path__
```

VBA ( )Python

24.2

Office Excel (Access Word) VBA (Outlook office Office `PYTHONPATH` Pyhon *Config*

CHAPTER 25

xlwings R Julia

While xlwings is a pure Python package, there are cross-language packages that allow for a relatively straightforward use from/with other languages. This means, however, that you'll always need to have Python with xlwings installed in addition to R or Julia. We recommend the Anaconda distribution, see also .

25.1 R

R Windows Mac R (UDF) (`RunPython` , *"RunPython" Python*) :

- R Python
- `R_HOME` `C:\Program Files\R\R-x.x.x`
- `R_USER` `C:\Users\<user>`
- `C:\Program Files\R\R-x.x.x\bin` PATH
- windows(!)

25.1.1 R

Excel R (`r_file.R`):

```
myfunction <- function(x, y){
    return(x * y)
}
```

Python   :

```python
import xlwings as xw
import rpy2.robjects as robjects
# you might want to use some relative path or place the file in R's current␣
↪working dir
robjects.r.source(r"C:\path\to\r_file.R")

@xw.func
def myfunction(x, y):
    myfunc = robjects.r['myfunction']
    return tuple(myfunc(x, y))
```

(: *User Defined Functions (UDFs)*),   Excel   UDF

25.1.2 R

Excel   R   (r_file.R):

```r
array_function <- function(m1, m2){
  # Matrix multiplication
  return(m1 %*% m2)
}
```

Python   :

```python
import xlwings as xw
import numpy as np
import rpy2.robjects as robjects
from rpy2.robjects import numpy2ri

robjects.r.source(r"C:\path\to\r_file.R")
numpy2ri.activate()

@xw.func
@xw.arg("x", np.array, ndim=2)
@xw.arg("y", np.array, ndim=2)
def array_function(x, y):
    array_func = robjects.r['array_function']
    return np.array(array_func(x, y))
```

(: *User Defined Functions (UDFs)*),   Excel   UDF

25.2 Julia

:

- Julia Python
- Julia Pkg.add("PyCall")

Julia xlwings( ):

```
julia> using PyCall
julia> @pyimport xlwings as xw

julia> xw.Book()
PyObject <Book [Workbook1]>

julia> xw.Range("A1")[:value] = "Hello World"
julia> xw.Range("A1")[:value]
"Hello World"
```

CHAPTER 26

Python API

26.1

`xlwings.view(obj, sheet=None, table=True)`

- **obj** (*any type with built-in converter*) – numpy arrays pandas dataframes
- **sheet** (Sheet, *default None*) –
- **table** (*bool, default True*) – If your object is a pandas DataFrame, by default it is formatted as an Excel Table

```
>>> import xlwings as xw
>>> import pandas as pd
>>> import numpy as np
>>> df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])
>>> xw.view(df)
```

See also: *load*

0.22.0 .

`xlwings.load(index=1, header=1)`

Loads the selected cell(s) of the active workbook into a pandas DataFrame. If you select a single cell that has adjacent cells, the range is auto-expanded and turned into a pandas DataFrame. If you don't have pandas installed, it returns the values as nested lists.

- **index** (*bool or int, default 1*) – Defines the number of columns on the left that will be turned into the DataFrame's index
- **header** (*bool or int, default 1*) – Defines the number of rows at the top that will be turned into the DataFrame's columns

```
>>> import xlwings as xw
>>> xw.load()
```

See also: *view*

0.22.0 .

26.2

26.2.1 Apps

class xlwings.main.Apps(*impl*)

app :

```
>>> import xlwings as xw
>>> xw.apps
Apps([<Excel App 1668>, <Excel App 1644>])
```

active

app

0.9.0 .

add()

App App app App

count

app

0.9.0 .

keys()

PID App Excel

0.13.0 .

26.2.2 App

class xlwings.**App**(*visible=None*, *spec=None*, *add_book=True*, *impl=None*)

app Excel Excel :

```
>>> import xlwings as xw
>>> app1 = xw.App()
>>> app2 = xw.App()
```

app apps :

```
>>> xw.apps
Apps([<Excel App 1668>, <Excel App 1644>])
>>> xw.apps[1668]  # get the available PIDs via xw.apps.keys()
<Excel App 1668>
>>> xw.apps.active
<Excel App 1668>
```

- **visible** (*bool, default None*) – ap- p visible=True
- **spec** (*str, default None*) – Mac , Excel `/Applications/Microsoft Office 2011/Microsoft Excel` `/Applications/Microsoft Excel`` Windows , xlwings Excel `` >` Office

: Mac , xlwings Excel Mac Excel Windows Excel Ex

activate(*steal_focus=False*)

Excel

steal_focus (*bool, default False*) – True, Ex- cel Python Excel

0.9.0 .

api

(pywin32 appscript)

0.9.0 .

books

0.9.0 .

calculate()

0.3.6 .

calculation

calculation : 'manual'( ) , 'automatic'( ) , 'semiautomatic'( )

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> wb.app.calculation = 'manual'
```

0.9.0 .

display_alerts

True False Excel

0.9.0 .

hwnd

Window (Windows)

0.9.0 .

kill()

Excel app

0.9.0 .

macro(*name*)

Excel VBA (sub)

name (Name of Sub or Function with or without module name, e.g. 'Module1.MyMacro' or 'MyMacro') –

VBA :

```
Function MySum(x, y)
    MySum = x + y
End Function
```

:

```
>>> import xlwings as xw
>>> app = xw.App()
>>> my_sum = app.macro('MySum')
>>> my_sum(1, 2)
3
```

: *Book.macro()*

0.9.0 .

pid
app PID

0.9.0 .

quit()

0.3.3 .

range(*cell1*, *cell2=None*)
, *Range()*

0.9.0 .

screen_updating
(False) screen_updating True

0.3.3 .

selection

0.9.0 .

startup_path
Returns the path to `XLSTART` which is where the xlwings add-in gets copied to by doing `xlwings addin install`.

0.19.4 .

status_bar
Gets or sets the value of the status bar. Returns `False` if Excel has control of it.

0.20.0 .

version
Excel

```
>>> import xlwings as xw
>>> xw.App().version
VersionNumber('15.24')
>>> xw.apps[10559].version.major
15
```

0.9.0 .

visible
Excel visible True False

0.3.3 .

26.2.3 Books

class xlwings.main.Books(*impl*)

:

```
>>> import xlwings as xw
>>> xw.books  # active app
Books([<Book [Book1]>, <Book [Book2]>])
>>> xw.apps[10559].books  # specific app, get the PIDs via xw.apps.keys()
Books([<Book [Book1]>, <Book [Book2]>])
```

0.9.0 .

active

add()

open(*fullname, update_links=None, read_only=None, format=None, password=None, write_res_password=None, ignore_read_only_recommended=None, origin=None, delimiter=None, editable=None, notify=None, converter=None, add_to_mru=None, local=None, corrupt_load=None*)

- **fullname** (*str or path-like object*) – `r'C:\path\to\file.xlsx'` `'file.xlsm'`
- **Parameters** (*Other*) – *xlwings.Book()*

Book

Book that has been opened.

26.2.4 Book

class xlwings.Book(*fullname=None, update_links=None, read_only=None, format=None, password=None, write_res_password=None, ignore_read_only_recommended=None, origin=None, delimiter=None, editable=None, notify=None, converter=None, add_to_mru=None, local=None, corrupt_load=None, impl=None*)

book books :

```
>>> import xlwings as xw
>>> xw.books[0]
<Book [Book1]>
```

`xw.Book` : app book book app `xw.books` app, :

```
>>> app = xw.App()  # or something like xw.apps[10559] for existing apps,␣
↪get the PIDs via xw.apps.keys()
>>> app.books['Book1']
```

	xw.Book	xw.books
	`xw.Book()`	`xw.books.add()`
	`xw.Book('Book1')`	`xw.books['Book1']`
	`xw.Book(r'C:/path/to/file.xlsx')`	`xw.books.open(r'C:/path/to/file.xlsx')`

- **`fullname`** (*str or path-like object, default None*) – (`xlsx` , `xlsm`)
- **`update_links`** (*bool, default None*) –
- **`read_only`** (*bool, default False*) – `True`
- **`format`** (*str*) –
- **`password`** (*str*) –
- **`write_res_password`** (*str*) –
- **`ignore_read_only_recommended`** (*bool, default False*) – `True`
- **`origin`** (*int*) – XlPlatform [1 Mac 2 Windows 3 Dos]
- **`delimiter`** (*str*) – format 6
- **`editable`** (*bool, default False*) – Excel4.0
- **`notify`** (*bool, default False*) –
- **`converter`** (*int*) –
- **`add_to_mru`** (*bool, default False*) –
- **`local`** (*bool, default False*) – If `True`, saves files against the language of Excel, otherwise against the language of VBA. Not supported on macOS.
- **`corrupt_load`** (*int, default xlNormalLoad*) – xlNormalLoad xlRepairFile xlExtractData macOS

`activate`(*steal_focus=False*)

`steal_focus` (*bool, default False*) – True, Python Excel

`api`

(`pywin32` `appscript`)

0.9.0 .

app

app

0.9.0 .

classmethod caller()

References the calling book when the Python function is called from Excel via `RunPython`. Pack it into the function being called from Excel, e.g.:

```python
import xlwings as xw

 def my_macro():
    wb = xw.Book.caller()
    wb.sheets[0].range('A1').value = 1
```

Python , `xw.Book.set_mock_caller()`

0.3.0 .

close()

0.1.1 .

fullname

macro(*name*)

Excel VBA

name (Name of Sub or Function with or without module name, e.g. `'Module1.MyMacro'` or `'MyMacro'`) –

VBA :

```vb
Function MySum(x, y)
    MySum = x + y
End Function
```

:

```python
>>> import xlwings as xw
>>> wb = xw.books.active
>>> my_sum = wb.macro('MySum')
>>> my_sum(1, 2)
3
```

App.macro()

0.7.1 .

name

names

0.9.0 .

save(*path=None*)

Excel SaveAs()

path (*str or path-like object, default None*) –

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> wb.save()
>>> wb.save(r'C:\path\to\new_file_name.xlsx')
```

0.3.1 .

selection

0.9.0 .

set_mock_caller()

Pyhton Excel RunPyton xw.Book.caller()

```
# This code runs unchanged from Excel via RunPython and from Python␣
↪directly
import os
import xlwings as xw

def my_macro():
    sht = xw.Book.caller().sheets[0]
    sht.range('A1').value = 'Hello xlwings!'

if __name__ == '__main__':
    xw.Book('file.xlsm').set_mock_caller()
    my_macro()
```

0.3.1 .

sheets

0.9.0 .

to_pdf(*path=None*, *include=None*, *exclude=None*)

Exports the whole Excel workbook or a subset of the sheets to a PDF file. If you want to print hidden sheets, you will need to list them explicitely under `include`.

- **path** (*str or path-like object, default None*) – Path to the PDF file, defaults to the same name as the workbook, in the same directory. For unsaved workbooks, it defaults to the current working directory instead.
- **include** (*int or str or list, default None*) – Which sheets to include: provide a selection of sheets in the form of sheet indices (1-based like in Excel) or sheet names. Can be an int/str for a single sheet or a list of int/str for multiple sheets.
- **exclude** (*int or str or list, default None*) – Which sheets to exclude: provide a selection of sheets in the form of sheet indices (1-based like in Excel) or sheet names. Can be an int/str for a single sheet or a list of int/str for multiple sheets.

```
>>> wb = xw.Book()
>>> wb.sheets[0]['A1'].value = 'PDF'
>>> wb.to_pdf()
```

See also *xlwings.Sheet.to_pdf()*

0.21.1 .

26.2.5 Sheets

class xlwings.main.**Sheets**(*impl*)

sheet :

```
>>> import xlwings as xw
>>> xw.sheets  # active book
Sheets([<Sheet [Book1]Sheet1>, <Sheet [Book1]Sheet2>])
>>> xw.Book('Book1').sheets  # specific book
Sheets([<Sheet [Book1]Sheet1>, <Sheet [Book1]Sheet2>])
```

0.9.0 .

active

(Sheet)

add(*name=None*, *before=None*, *after=None*)

- **name** (*str, default None*) – Excel
- **before** (Sheet*, default None*) –
- **after** (Sheet*, default None*) –

26.2.6 Sheet

class xlwings.Sheet(*sheet=None, impl=None*)

sheet sheets :

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> wb.sheets[0]
<Sheet [Book1]Sheet1>
>>> wb.sheets['Sheet1']
<Sheet [Book1]Sheet1>
>>> wb.sheets.add()
<Sheet [Book1]Sheet2>
```

0.9.0 .

activate()

sheet

api

(pywin32 appscript)

0.9.0 .

autofit(*axis=None*)

axis (*string, default None*) –

- rows r
- columns c
-

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> wb.sheets['Sheet1'].autofit('c')
>>> wb.sheets['Sheet1'].autofit('r')
>>> wb.sheets['Sheet1'].autofit()
```

0.2.3 .

book

cells

( )

0.9.0 .

charts

Charts

0.9.0 .

clear()

clear_contents()

copy(*before=None*, *after=None*, *name=None*)

Copy a sheet to the current or a new Book. By default, it places the copied sheet after all existing sheets in the current Book. Returns the copied sheet.

0.22.0 .

- **before** (*sheet object, default None*) – The sheet object before which you want to place the sheet
- **after** (*sheet object, default None*) – The sheet object after which you want to place the sheet, by default it is placed after all existing sheets
- **name** (*str, default None*) – The sheet name of the copy

Sheet object – The copied sheet

Sheet

```
# Create two books and add a value to the first sheet of the first book
first_book = xw.Book()
second_book = xw.Book()
first_book.sheets[0]['A1'].value = 'some value'

# Copy to same Book with the default location and name
first_book.sheets[0].copy()

# Copy to same Book with custom sheet name
first_book.sheets[0].copy(name='copied')
```

( )

( )

```
# Copy to second Book requires to use before or after
first_book.sheets[0].copy(after=second_book.sheets[0])
```

delete()

index

(Excel 1)

name

names

("SheetName!" (!))

0.9.0 .

pictures

Pictures

0.9.0 .

range(*cell1*, *cell2=None*)

, *Range()*

0.9.0 .

render_template(***data*)

This method requires xlwings *PRO*.

Replaces all Jinja variables (e.g `{{ myvar }}`) in the sheet with the keyword argument that has the same name. Following variable types are supported:

strings, numbers, lists, simple dicts, NumPy arrays, Pandas DataFrames, PIL Image objects that have a filename and Matplotlib figures.

0.22.0 .

data (*kwargs*) – All key/value pairs that are used in the template.

sheet

xlwings Sheet

```
>>> import xlwings as xw
>>> book = xw.Book()
>>> book.sheets[0]['A1:A2'].value = '{{ myvar }}'
>>> book.sheets[0].render_template(myvar='test')
```

See also `xlwings.pro.reports.create_report()`

select()

0.9.0 .

shapes

Shapes

0.9.0 .

tables

See *Tables*

0.21.0 .

to_pdf(*path=None*)

Exports the sheet to a PDF file.

path (*str or path-like object, default None*) – Path to the PDF file, defaults to the name of the sheet in the same directory of the workbook. For unsaved workbooks, it defaults to the current working directory instead.

```
>>> wb = xw.Book()
>>> sheet = wb.sheets[0]
>>> sheet['A1'].value = 'PDF'
>>> sheet.to_pdf()
```

See also *xlwings.Book.to_pdf()*

0.22.3 .

used_range

xw.Range

0.13.0 .

visible

Gets or sets the visibility of the Sheet (bool).

0.21.1 .

26.2.7 Range

class xlwings.Range(*cell1=None*, *cell2=None*, ***options*)

- **cell1** (*str or tuple or Range*) – A1 xw.Range ('A1:B2')
- **cell2** (*str or tuple or Range, default None*) – A1 xw.Range

:

```
import xlwings as xw
xw.Range('A1')
xw.Range('A1:C3')
xw.Range((1,1))
xw.Range((1,1), (3,3))
xw.Range('NamedRange')
xw.Range(xw.Range('A1'), xw.Range('B2'))
```

:

```
xw.books['MyBook.xlsx'].sheets[0].range('A1')
```

add_hyperlink(*address*, *text_to_display=None*, *screen_tip=None*)
( )

- **address** (*str*) –
- **text_to_display** (*str, default None*) –
- **screen_tip** (*str, default None*) – '<address> - '

0.3.0 .

address

get_address()

0.9.0 .

api

(pywin32 appscript)

0.9.0 .

autofit()

- xw.Range('A1:B2').columns.autofit()
- xw.Range('A1:B2').rows.autofit()

0.9.0 .

`clear()`

`clear_contents()`

`color`

RGB (0, 0, 0) `None` ,

RGB

tuple

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> xw.Range('A1').color = (255,255,255)
>>> xw.Range('A2').color
(255, 255, 255)
>>> xw.Range('A2').color = None
>>> xw.Range('A2').color is None
True
```

0.3.0 .

`column`

Integer

0.3.5 .

`column_width`

( ) `Normal` 0(0)

`None`

: 0 <= <= 255

:

float

0.4.0 .

`columns`

`RangeColumns`

0.9.0 .

copy(*destination=None*)

destination (xlwings.Range) – xlwings Range

None

count

current_region

( ) Windows `Ctrl-*` Mac `shift-Ctrl-Space`

Range object

delete(*shift=None*)

shift (*str, default None*) – `left` `up` Excel

None

end(*direction*)

`Ctrl+Up` , `Ctrl+down` , `Ctrl+left` , `Ctrl+right`

direction (*One of 'up', 'down', 'right', 'left'*) –

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> xw.Range('A1:B2').value = 1
>>> xw.Range('A1').end('down')
<Range [Book1]Sheet1!$A$2>
>>> xw.Range('B2').end('right')
<Range [Book1]Sheet1!$B$2>
```

0.9.0 .

expand(*mode='table'*)

(`Range.end()`).

mode (*str, default 'table'*) – `'down'` , `'right'` ,"'table'" (=down + right)

Range

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> xw.Range('A1').value = [[None, 1], [2, 3]]
>>> xw.Range('A1').expand().address
$A$1:$B$2
>>> xw.Range('A1').expand('right').address
$A$1:$B$1
```

0.9.0 .

formula

formula2

Gets or sets the formula2 for the given Range.

formula_array

0.7.1 .

get_address(*row_absolute=True, column_absolute=True, include_sheetname=False, external=False*)

address

- **row_absolute** (*bool, default True*) – True
- **column_absolute** (*bool, default True*) – True
- **include_sheetname** (*bool, default False*) – True external=True
- **external** (*bool, default False*) – True

str

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> xw.Range((1,1)).get_address()
'$A$1'
>>> xw.Range((1,1)).get_address(False, False)
'A1'
>>> xw.Range((1,1), (3,3)).get_address(True, False, True)
'Sheet1!A$1:C$3'
```

( )

( )

```
>>> xw.Range((1,1), (3,3)).get_address(True, False, external=True)
'[Book1]Sheet1!A$1:C$3'
```

0.2.3 .

has_array
Are we part of an Array formula?

height

float

0.4.0 .

hyperlink
( )

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> xw.Range('A1').value
'www.xlwings.org'
>>> xw.Range('A1').hyperlink
'http://www.xlwings.org'
```

0.3.0 .

insert(*shift=None*, *copy_origin='format_from_left_or_above'*)

- **shift** (*str, default None*) – right or down Excel
- **copy_origin** (*str, default format_from_left_or_above*) – format_from_left_or_above format_from_right_or_below macOS

None

last_cell

Range

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> rng = xw.Range('A1:E4')
>>> rng.last_cell.row, rng.last_cell.column
(4, 5)
```

0.3.5 .

left

A (point)

float

0.6.0 .

merge(*across=False*)

Creates a merged cell from the specified Range object.

across (*bool, default False*) – True to merge cells in each row of the specified Range as separate merged cells.

merge_area

Returns a Range object that represents the merged Range containing the specified cell. If the specified cell isn't in a merged range, this property returns the specified cell.

merge_cells

Returns `True` if the Range contains merged cells, otherwise `False`

name

0.4.0 .

number_format

(`number_format`)

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> xw.Range('A1').number_format
'General'
>>> xw.Range('A1:C3').number_format = '0.00%'
>>> xw.Range('A1:C3').number_format
'0.00%'
```

0.2.3 .

offset(*row_offset=0*, *column_offset=0*)

Range

0.3.0 .

options(*convert=None*, ***options*)

Excel (base converter)

convert (*object, default None*) – dict, np.array, pd.DataFrame, pd.Series ,

- **ndim** (*int, default None*) –
- **numbers** (type*, default None*) – int
- **dates** (type*, default None*) – datetime.date datetime.datetime
- **empty** (*object, default None*) –
- **transpose** (*Boolean, default False*) –
- **expand** (*str, default None*) – 'table' , 'down' , 'right' =>

Range object

0.7.0 .

paste(*paste=None*, *operation=None*, *skip_blanks=False*, *transpose=False*)

- **paste** (*str, default None*) – all_merging_conditional_formats, all, all_except_borders, all_using_source_theme, column_widths, comments, formats, formulas, formulas_and_number_formats, validation, values, values_and_number_formats.
- **operation** (*str, default None*) – “add”, “divide”, “multiply”, “subtract”
- **skip_blanks** (*bool, default False*) – True
- **transpose** (*bool, default False*) – True

None

raw_value

xlwings (pywin32 appscript)

resize(*row_size=None*, *column_size=None*)

- **row_size** (*int > 0*) – (None ,)
- **column_size** (*int > 0*) – (None ,)

Range

0.3.0 .

row

Integer

0.3.5 .

row_height

`point` `None`

row_height : 0 <= row_height <= 409.5

float

0.4.0 .

rows

RangeRows

0.9.0 .

select()

0.9.0 .

shape

0.3.0 .

sheet

0.9.0 .

size

0.3.0 .

`table`

Returns a Table object if the range is part of one, otherwise `None`.

0.21.0 .

`top`

`point`

float

0.6.0 .

`unmerge()`

Separates a merged area into individual cells.

`value`

: *xlwings.Range.options()*

`width`

`point`

float

0.4.0 .

26.2.8 RangeRows

`class xlwings.RangeRows(rng)`

Range.rows

```
import xlwings as xw

rng = xw.Range('A1:C4')

assert len(rng.rows) == 4  # or rng.rows.count

rng.rows[0].value = 'a'

assert rng.rows[2] == xw.Range('A3:C3')
assert rng.rows(2) == xw.Range('A2:C2')
```

( )

( )

```
for r in rng.rows:
    print(r.address)
```

autofit()

count

0.9.0 .

26.2.9 RangeColumns

class xlwings.**RangeColumns**(*rng*)

Range.columns

```
import xlwings as xw

rng = xw.Range('A1:C4')

assert len(rng.columns) == 3  # or rng.columns.count

rng.columns[0].value = 'a'

assert rng.columns[2] == xw.Range('C1:C4')
assert rng.columns(2) == xw.Range('B1:B4')

for c in rng.columns:
    print(c.address)
```

autofit()

count

0.9.0 .

26.2.10 Shapes

class xlwings.main.**Shapes**(*impl*)

(shape) :

```
>>> import xlwings as xw
>>> xw.books['Book1'].sheets[0].shapes
Shapes([<Shape 'Oval 1' in <Sheet [Book1]Sheet1>>, <Shape 'Rectangle 1' in
↪<Sheet [Book1]Sheet1>>])
```

0.9.0 .

api

(pywin32 appscript)

count

26.2.11 Shape

class xlwings.**Shape**(**args*, ***options*)

shapes :

```
>>> import xlwings as xw
>>> sht = xw.books['Book1'].sheets[0]
>>> sht.shapes[0]  # or sht.shapes['ShapeName']
<Shape 'Rectangle 1' in <Sheet [Book1]Sheet1>>
```

0.9.0 .

activate()

(shape)

0.5.0 .

api

(pywin32 appscript)

0.19.2 .

delete()

0.5.0 .

height

point

0.5.0 .

left

point

0.5.0 .

name

0.5.0 .

parent

0.9.0 .

scale_height(*factor*, *relative_to_original_size=False*, *scale='scale_from_top_left'*)

factor [float] For example 1.5 to scale it up to 150%

relative_to_original_size [bool, optional] If `False`, it scales relative to current height (default). For `True` must be a picture or OLE object.

scale [str, optional] One of `scale_from_top_left` (default), `scale_from_bottom_right`, `scale_from_middle`

0.19.2 .

scale_width(*factor*, *relative_to_original_size=False*, *scale='scale_from_top_left'*)

factor [float] For example 1.5 to scale it up to 150%

relative_to_original_size [bool, optional] If `False`, it scales relative to current width (default). For `True` must be a picture or OLE object.

scale [str, optional] One of `scale_from_top_left` (default), `scale_from_bottom_right`, `scale_from_middle`

0.19.2 .

text

Returns or sets the text of a shape.

0.21.4 .

top

`point`

0.5.0 .

type

0.9.0 .

width

`point`

0.5.0 .

26.2.12 Charts

`class xlwings.main.Charts(impl)`

(`chart`) :

```
>>> import xlwings as xw
>>> xw.books['Book1'].sheets[0].charts
Charts([<Chart 'Chart 1' in <Sheet [Book1]Sheet1>>, <Chart 'Chart 1' in
↪<Sheet [Book1]Sheet1>>])
```

0.9.0 .

add(*left=0*, *top=0*, *width=355*, *height=211*)

- **left** (*float, default 0*) – point
- **top** (*float, default 0*) – point
- **width** (*float, default 355*) – point
- **height** (*float, default 211*) – point

Chart

```
>>> import xlwings as xw
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = [['Foo1', 'Foo2'], [1, 2]]
>>> chart = sht.charts.add()
>>> chart.set_source_data(sht.range('A1').expand())
>>> chart.chart_type = 'line'
>>> chart.name
'Chart1'
```

api

(pywin32 appscript)

count

26.2.13 Chart

class xlwings.**Chart**(*name_or_index=None*, *impl=None*)

chart *charts* :

```
>>> import xlwings as xw
>>> sht = xw.books['Book1'].sheets[0]
>>> sht.charts[0]  # or sht.charts['ChartName']
<Chart 'Chart 1' in <Sheet [Book1]Sheet1>>
```

api

(pywin32 appscript)

0.9.0 .

chart_type

Returns and sets the chart type of the chart. The following chart types are available:

`3d_area`, `3d_area_stacked`, `3d_area_stacked_100`, `3d_bar_clustered`, `3d_bar_stacked`, `3d_bar_stacked_100`, `3d_column`, `3d_column_clustered`, `3d_column_stacked`, `3d_column_stacked_100`, `3d_line`, `3d_pie`, `3d_pie_exploded`, `area`, `area_stacked`, `area_stacked_100`, `bar_clustered`, `bar_of_pie`, `bar_stacked`, `bar_stacked_100`, `bubble`, `bubble_3d_effect`, `column_clustered`, `column_stacked`, `column_stacked_100`, `combination`, `cone_bar_clustered`, `cone_bar_stacked`, `cone_bar_stacked_100`, `cone_col`, `cone_col_clustered`, `cone_col_stacked`, `cone_col_stacked_100`, `cylinder_bar_clustered`, `cylinder_bar_stacked`, `cylinder_bar_stacked_100`, `cylinder_col`, `cylinder_col_clustered`, `cylinder_col_stacked`, `cylinder_col_stacked_100`, `doughnut`, `doughnut_exploded`, `line`, `line_markers`, `line_markers_stacked`, `line_markers_stacked_100`, `line_stacked`, `line_stacked_100`, `pie`, `pie_exploded`, `pie_of_pie`, `pyramid_bar_clustered`, `pyramid_bar_stacked`, `pyramid_bar_stacked_100`, `pyramid_col`, `pyramid_col_clustered`, `pyramid_col_stacked`, `pyramid_col_stacked_100`, `radar`, `radar_filled`, `radar_markers`, `stock_hlc`, `stock_ohlc`, `stock_vhlc`, `stock_vohlc`, `surface`, `surface_top_view`, `surface_top_view_wireframe`, `surface_wireframe`, `xy_scatter`, `xy_scatter_lines`, `xy_scatter_lines_no_markers`, `xy_scatter_smooth`, `xy_scatter_smooth_no_markers`

0.1.1 .

delete()

height

point

left

point

name

parent

0.9.0 .

set_source_data(*source*)

source (Range) – `xw.books['Book1'].sheets[0].range('A1')`

top

point

width
point

26.2.14 Pictures

class xlwings.main.**Pictures**(*impl*)
(picture) :

```
>>> import xlwings as xw
>>> xw.books['Book1'].sheets[0].pictures
Pictures([<Picture 'Picture 1' in <Sheet [Book1]Sheet1>>, <Picture 'Picture␣
↪2' in <Sheet [Book1]Sheet1>>])
```

0.9.0 .

add(*image, link_to_file=False, save_with_document=True, left=0, top=0, width=None, height=None, name=None, update=False, scale=1*)

- **image** (*str or path-like object or matplotlib.figure.Figure*) – Matplotlib
- **left** (*float, default 0*) – ( ) point
- **top** (*float, default 0*) – ( ) point
- **width** (*float, default None*) – point PIL/Pillow , 100
- **height** (*float, default None*) – point PIL/Pillow , 100
- **name** (*str, default None*) – Excel Excel 'Picture 1'
- **update** (*bool, default False*) –

Picture

1. Picture

```
>>> import xlwings as xw
>>> sht = xw.Book().sheets[0]
>>> sht.pictures.add(r'C:\path\to\file.jpg')
<Picture 'Picture 1' in <Sheet [Book1]Sheet1>>
```

2. Matplotlib

```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> plt.plot([1, 2, 3, 4, 5])
>>> sht.pictures.add(fig, name='MyPlot', update=True)
<Picture 'MyPlot' in <Sheet [Book1]Sheet1>>
```

api

(pywin32 appscript)

count

26.2.15 Picture

class xlwings.**Picture**(*impl=None*)

pictures

```
>>> import xlwings as xw
>>> sht = xw.books['Book1'].sheets[0]
>>> sht.pictures[0]  # or sht.charts['PictureName']
<Picture 'Picture 1' in <Sheet [Book1]Sheet1>>
```

0.9.0 .

api

(pywin32 appscript)

0.9.0 .

delete()

0.5.0 .

height

point

0.5.0 .

left

point

0.5.0 .

name

0.5.0 .

parent

0.9.0 .

top

point

0.5.0 .

update(*image*)

image (*str or path-like object or matplotlib.figure.Figure*) – Matplotlib

0.5.0 .

width

point

0.5.0 .

26.2.16 Names

class xlwings.main.**Names**(*impl*)

(name) :

```
>>> import xlwings as xw
>>> sht = xw.books['Book1'].sheets[0]
>>> sht.names
[<Name 'MyName': =Sheet1!$A$3>]
```

0.9.0 .

add(*name*, *refers_to*)

- **name** (*str*) –
- **refers_to** (*str*) – A1

Name

0.9.0 .

api

(pywin32 appscript)

0.9.0 .

count

26.2.17 Name

`class xlwings.Name(`*impl*`)`

name names :

```
>>> import xlwings as xw
>>> sht = xw.books['Book1'].sheets[0]
>>> sht.names[0]  # or sht.names['MyName']
<Name 'MyName': =Sheet1!$A$3>
```

0.9.0 .

api

(pywin32 appscript)

0.9.0 .

delete()

0.9.0 .

name

0.9.0 .

refers_to

A1

0.9.0 .

refers_to_range

0.9.0 .

26.2.18 Tables

`class xlwings.main.Tables(`*impl*`)`

A collection of all *table* objects on the specified sheet:

```
>>> import xlwings as xw
>>> xw.books['Book1'].sheets[0].tables
Tables([<Table 'Table1' in <Sheet [Book11]Sheet1>>, <Table 'Table2' in
↪<Sheet [Book11]Sheet1>>])
```

0.21.0 .

add(*source=None, name=None, source_type=None, link_source=None, has_headers=True, destination=None, table_style_name='TableStyleMedium2'***)**

Creates a Table to the specified sheet.

- **source** (*xlwings range, default None*) – An xlwings range object, representing the data source.
- **name** (*str, default None*) – The name of the Table. By default, it uses the autogenerated name that is assigned by Excel.
- **source_type** (*str, default None*) – This currently defaults to `xlSrcRange`, i.e. expects an xlwings range object. No other options are allowed at the moment.
- **link_source** (*bool, default None*) – Currently not implemented as this is only in case `source_type` is `xlSrcExternal`.
- **has_headers** (*bool or str, default True*) – Indicates whether the data being imported has column labels. Defaults to `True`. Possible values: `True`, `FAlse`, `'guess'`
- **destination** (*xlwings range, default None*) – Currently not implemented as this is used in case `source_type` is `xlSrcExternal`.
- **table_style_name** (*str, default 'TableStyleMedium2'*) – Possible strings: `'TableStyleLightN''` (where N is 1-21), `'TableStyleMediumN'` (where N is 1-28), `'TableStyleDarkN'` (where N is 1-11)

Table

```
>>> import xlwings as xw
>>> sheet = xw.Book().sheets[0]
>>> sheet['A1'].value = [['a', 'b'], [1, 2]]
>>> table = sheet.tables.add(source=sheet['A1'].expand(), name='MyTable
↪')
>>> table
<Table 'MyTable' in <Sheet [Book1]Sheet1>>
```

26.2.19 Table

class xlwings.main.**Table**(**args*, ***options*)

The table object is a member of the *tables* collection:

```
>>> import xlwings as xw
>>> sht = xw.books['Book1'].sheets[0]
>>> sht.tables[0]  # or sht.tables['TableName']
<Table 'Table 1' in <Sheet [Book1]Sheet1>>
```

0.21.0 .

api

(pywin32 appscript)

data_body_range

Returns an xlwings range object that represents the range of values, excluding the header row

display_name

Returns or sets the display name for the specified Table object

header_row_range

Returns an xlwings range object that represents the range of the header row

insert_row_range

Returns an xlwings range object representing the row where data is going to be inserted. This is only available for empty tables, otherwise it'll return `None`

name

Returns or sets the name of the Table.

parent

Returns the parent of the table.

range

Returns an xlwings range object of the table.

show_autofilter

Turn the autofilter on or off by setting it to `True` or `False` (read/write boolean)

show_headers

Show or hide the header (read/write)

show_table_style_column_stripes

Returns or sets if the Column Stripes table style is used for (read/write boolean)

show_table_style_first_column

Returns or sets if the first column is formatted (read/write boolean)

show_table_style_last_column

Returns or sets if the last column is displayed (read/write boolean)

show_table_style_row_stripes

Returns or sets if the Row Stripes table style is used (read/write boolean)

show_totals

Gets or sets a boolean to show/hide the Total row.

table_style

Gets or sets the table style. See *`Tables.add`* for possible values.

totals_row_range

Returns an xlwings range object representing the Total row

update(*data*)

This method requires xlwings *PRO*

Updates the Excel table with the provided data. Currently restricted to DataFrames.

0.21.3 .

data (*pandas DataFrame*) – Currently restricted to pandas DataFrames. If you want to hide the index, set the first column as the index, e.g. `df.set_index('column_name')`.

Table

```
import pandas as pd
import xlwings as xw

sheet = xw.Book('Book1.xlsx').sheets[0]
table_name = 'mytable'

# Sample DataFrame
nrows, ncols = 3, 3
df = pd.DataFrame(data=nrows * [ncols * ['test']],
                  columns=['col ' + str(i) for i in range(ncols)])

# Hide the index, then insert a new table if it doesn't exist yet,
# otherwise update the existing one
df = df.set_index('col 0')
if table_name in [table.name for table in sheet.tables]:
    sheet.tables[table_name].update(df)
else:
    mytable = sheet.tables.add(source=sheet['A1'], name=table_name).
↪update(df)
```

26.2.20 Font

class `xlwings.main.`**`Font`**(*impl*)

The font object can be accessed as an attribute of the range or shape object.

- `mysheet['A1'].font`
- `mysheet.shapes[0].font`

0.23.0 .

api

(pywin32 appscript)

0.23.0 .

bold

Returns or sets the bold property (boolean).

```
>>> sheet['A1'].font.bold = True
>>> sheet['A1'].font.bold
True
```

0.23.0　.

color
Returns or sets the color property (tuple).

```
>>> sheet['A1'].font.color = (255, 0, 0) # RGB tuple
>>> sheet['A1'].font.color
(255, 0, 0)
```

0.23.0　.

italic
Returns or sets the italic property (boolean).

```
>>> sheet['A1'].font.italic = True
>>> sheet['A1'].font.italic
True
```

0.23.0　.

name
Returns or sets the name of the font (str).

```
>>> sheet['A1'].font.name = 'Calibri'
>>> sheet['A1'].font.name
Calibri
```

0.23.0　.

size
Returns or sets the size (float).

```
>>> sheet['A1'].font.size = 13
>>> sheet['A1'].font.size
13
```

0.23.0　.

26.2.21 Characters

`class xlwings.main.`**`Characters`**`(`*impl*`)`
The characters object can be accessed as an attribute of the range or shape object.

- `mysheet['A1'].characters`
- `mysheet.shapes[0].characters`

: On macOS, `characters` are currently not supported due to bugs/lack of support in AppleScript.

0.23.0 .

api

(`pywin32` `appscript`)

0.23.0 .

font

Returns or sets the text property of a `characters` object.

```
>>> sheet['A1'].characters[1:3].font.bold = True
>>> sheet['A1'].characters[1:3].font.bold
True
```

0.23.0 .

text

Returns or sets the text property of a `characters` object.

```
>>> sheet['A1'].value = 'Python'
>>> sheet['A1'].characters[:3].text
Pyt
```

0.23.0 .

26.2.22 Markdown

26.2.23 MarkdownStyle

26.3 UDF

`xlwings.func`(*category="xlwings", volatile=False, call_in_wizard=True*)

"Import Python UDFs" `xlwings.func` `Function( )` Excel

category [int or str, default "xlwings"] 1-14

0.10.3 .

volatile [bool, default False]

0.10.3 .

call_in_wizard [bool, default True] False

0.10.3 .

`xlwings.sub()`

`xlwings.sub` "Import Python UDFs" `Sub` ( ) Excel

`xlwings.arg(arg, convert=None, **options)`
Range.options()

x 2 numpy :

```
import xlwings as xw
import numpy as np

@xw.func
@xw.arg('x', np.array, ndim=2)
def add_one(x):
    return x + 1
```

`xlwings.ret(convert=None, **options)`
Range.options()

1) DataFrame :

```
import pandas as pd

@xw.func
@xw.ret(index=False, header=False)
def get_dataframe(n, m):
    return pd.DataFrame(np.arange(n * m).reshape((n, m)))
```

2)

: If your version of Excel supports the new native dynamic arrays, then you don't have to do anything special, and you shouldn't use the `expand` decorator! To check if your version of Excel supports it, see if you have the `=UNIQUE()` formula available. Native dynamic arrays were introduced in Office 365 Insider Fast at the end of September 2018.

`expand='table'` UDF `=TODAY()` ( )

Excel :

```
import xlwings as xw
import numpy as np

@xw.func
@xw.ret(expand='table')
def dynamic_array(n, m):
    return np.arange(n * m).reshape((n, m))
```

0.10.0 .

26.4 Reports

CHAPTER 27

REST API

0.13.0 .

27.1

xlwings offers an easy way to expose an Excel workbook via REST API both on Windows and macOS. This can be useful when you have a workbook running on a single computer and want to access it from another computer. Or you can build a Linux based web app that can interact with a legacy Excel application while you are in the progress of migrating the Excel functionality into your web app (if you need help with that, give us a shout).

REST API (Flask>=1.0, pip install Flask):

```
xlwings restapi run
```

Windows PowerShell Mac GET ("Book1") GET (Postman Insomnia REST API):

```
$ curl "http://127.0.0.1:5000/book/book1/sheets/0/range/A1:B2"
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 10.0,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
```

( )

( )

```
      "1",
      "2"
    ],
    [
      "3",
      "4"
    ]
  ],
  "formula_array": null,
  "height": 32.0,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
  "number_format": "General",
  "row": 1,
  "row_height": 16.0,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
    [
      1.0,
      2.0
    ],
    [
      3.0,
      4.0
    ]
  ],
  "width": 130.0
}
```

`Ctrl-C`

xlwings xlwings REST API *Python API* REST API Excel Excel REST API

: GET POST GitHub

27.2

`xlwings restapi run` Flask http://127.0.0.1:5000 `--host` `--port` Flask `FLASK_ENV=development`

Flask Flask docs

```
set FLASK_APP=xlwings.rest.api
flask run
```

Mac , `export FLASK_APP=xlwings.rest.api` `set FLASK_APP=xlwings.rest.api`

, WSGI HTTP gunicorn (Mac) waitress (Mac/Windows) API , gunicorn : `gunicorn xlwings.rest.api:api` waitress (API):

```
from xlwings.rest.api import api
from waitress import serve
serve(wsgiapp, host='127.0.0.1', port=5000)
```

27.3

Python API Python 0 (`xw.books[0]`) Excel 1 (`xw.books(1)`),REST API 0 , `/books/0`.

27.4

REST API *xlwings.Range.options()*

`/book/book1/sheets/0/range/A1?expand=table&transpose=true`

`options` `value`

27.5 Endpoint

Endpoint		
/book	*Book*	Excel workbook( ) workbook
/books	*Books*	Excel
/apps	*Apps*	Excel

27.6 Endpoint

27.6.1 /book

GET /book/<fullname_or_name>

:

```
{
  "app": 1104,
  "fullname": "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
  "name": "Book1.xlsx",
  "names": [
    "Sheet1!myname1",
    "myname2"
  ],
  "selection": "Sheet2!$A$1",
  "sheets": [
    "Sheet1",
    "Sheet2"
  ]
}
```

GET /book/<fullname_or_name>/names

:

```
{
  "names": [
    {
      "name": "Sheet1!myname1",
      "refers_to": "=Sheet1!$B$2:$C$3"
    },
    {
      "name": "myname2",
      "refers_to": "=Sheet1!$A$1"
    }
  ]
}
```

GET /book/<fullname_or_name>/names/<name>

:

```
{
  "name": "myname2",
  "refers_to": "=Sheet1!$A$1"
}
```

`GET /book/<fullname_or_name>/names/<name>/range`

:

```
{
  "address": "$A$1",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 1,
  "current_region": "$A$1:$B$2",
  "formula": "=1+1.1",
  "formula_array": "=1+1,1",
  "height": 14.25,
  "last_cell": "$A$1",
  "left": 0.0,
  "name": "myname2",
  "number_format": "General",
  "row": 1,
  "row_height": 14.3,
  "shape": [
    1,
    1
  ],
  "size": 1,
  "top": 0.0,
  "value": 2.1,
  "width": 51.0
}
```

`GET /book/<fullname_or_name>/sheets`

:

```
{
  "sheets": [
    {
      "charts": [
        "Chart 1"
      ],
      "name": "Sheet1",
      "names": [
        "Sheet1!myname1"
      ],
      "pictures": [
        "Picture 3"
      ],
      "shapes": [
```

( )

( )

```
        "Chart 1",
        "Picture 3"
      ],
      "used_range": "$A$1:$B$2"
    },
    {
      "charts": [],
      "name": "Sheet2",
      "names": [],
      "pictures": [],
      "shapes": [],
      "used_range": "$A$1"
    }
  ]
}
```

GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>

:

```
{
  "charts": [
    "Chart 1"
  ],
  "name": "Sheet1",
  "names": [
    "Sheet1!myname1"
  ],
  "pictures": [
    "Picture 3"
  ],
  "shapes": [
    "Chart 1",
    "Picture 3"
  ],
  "used_range": "$A$1:$B$2"
}
```

GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/charts

:

```
{
  "charts": [
    {
      "chart_type": "line",
      "height": 211.0,
```

( )

( )

```
      "left": 0.0,
      "name": "Chart 1",
      "top": 0.0,
      "width": 355.0
    }
  ]
}
```

GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/charts/<chart_name_or_ix>

:

```
{
  "chart_type": "line",
  "height": 211.0,
  "left": 0.0,
  "name": "Chart 1",
  "top": 0.0,
  "width": 355.0
}
```

GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/names

:

```
{
  "names": [
    {
      "name": "Sheet1!myname1",
      "refers_to": "=Sheet1!$B$2:$C$3"
    }
  ]
}
```

GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/names/<sheet_scope_name>

:

```
{
  "name": "Sheet1!myname1",
  "refers_to": "=Sheet1!$B$2:$C$3"
}
```

GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/names/<sheet_scope_name>/range

:

```
{
  "address": "$B$2:$C$3",
  "color": null,
  "column": 2,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "",
      ""
    ],
    [
      "",
      ""
    ]
  ],
  "formula_array": "",
  "height": 28.5,
  "last_cell": "$C$3",
  "left": 51.0,
  "name": "Sheet1!myname1",
  "number_format": "General",
  "row": 2,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 14.25,
  "value": [
    [
      null,
      null
    ],
    [
      null,
      null
    ]
  ],
  "width": 102.0
}
```

`GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/pictures`

:

```
{
  "pictures": [
    {
      "height": 100.0,
      "left": 0.0,
      "name": "Picture 3",
      "top": 0.0,
      "width": 100.0
    }
  ]
}
```

`GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/pictures/<picture_name_or_ix>`

:

```
{
  "height": 100.0,
  "left": 0.0,
  "name": "Picture 3",
  "top": 0.0,
  "width": 100.0
}
```

`GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/range`

:

```
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "=1+1.1",
      "a string"
    ],
    [
      "43395.0064583333",
      ""
    ]
  ],
  "formula_array": null,
  "height": 28.5,
  "last_cell": "$B$2",
```

( )

( )

```
  "left": 0.0,
  "name": null,
  "number_format": null,
  "row": 1,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
    [
      2.1,
      "a string"
    ],
    [
      "Mon, 22 Oct 2018 00:09:18 GMT",
      null
    ]
  ],
  "width": 102.0
}
```

GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/range/<address>

:

```
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "=1+1.1",
      "a string"
    ],
    [
      "43395.0064583333",
      ""
    ]
  ],
  "formula_array": null,
```

( )

( )

```
  "height": 28.5,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
  "number_format": null,
  "row": 1,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
    [
      2.1,
      "a string"
    ],
    [
      "Mon, 22 Oct 2018 00:09:18 GMT",
      null
    ]
  ],
  "width": 102.0
}
```

`GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/shapes`

:

```
{
  "shapes": [
    {
      "height": 211.0,
      "left": 0.0,
      "name": "Chart 1",
      "top": 0.0,
      "type": "chart",
      "width": 355.0
    },
    {
      "height": 100.0,
      "left": 0.0,
      "name": "Picture 3",
      "top": 0.0,
      "type": "picture",
```

( )

( )

```
      "width": 100.0
    }
  ]
}
```

`GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/shapes/<shape_name_or_ix>`

:

```
{
  "height": 211.0,
  "left": 0.0,
  "name": "Chart 1",
  "top": 0.0,
  "type": "chart",
  "width": 355.0
}
```

27.6.2 /books

`GET /books`

:

```
{
  "books": [
    {
      "app": 1104,
      "fullname": "Book1",
      "name": "Book1",
      "names": [],
      "selection": "Sheet2!$A$1",
      "sheets": [
        "Sheet1"
      ]
    },
    {
      "app": 1104,
      "fullname": "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
      "name": "Book1.xlsx",
      "names": [
        "Sheet1!myname1",
        "myname2"
      ],
      "selection": "Sheet2!$A$1",
      "sheets": [
```

( )

( )

```
        "Sheet1",
        "Sheet2"
      ]
    },
    {
      "app": 1104,
      "fullname": "Book4",
      "name": "Book4",
      "names": [],
      "selection": "Sheet2!$A$1",
      "sheets": [
        "Sheet1"
      ]
    }
  ]
}
```

GET /books/<book_name_or_ix>

:

```
{
  "app": 1104,
  "fullname": "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
  "name": "Book1.xlsx",
  "names": [
    "Sheet1!myname1",
    "myname2"
  ],
  "selection": "Sheet2!$A$1",
  "sheets": [
    "Sheet1",
    "Sheet2"
  ]
}
```

GET /books/<book_name_or_ix>/names

:

```
{
  "names": [
    {
      "name": "Sheet1!myname1",
      "refers_to": "=Sheet1!$B$2:$C$3"
    },
    {
```

( )

( )

```
      "name": "myname2",
      "refers_to": "=Sheet1!$A$1"
    }
  ]
}
```

GET /books/<book_name_or_ix>/names/<name>

:

```
{
  "name": "myname2",
  "refers_to": "=Sheet1!$A$1"
}
```

GET /books/<book_name_or_ix>/names/<name>/range

:

```
{
  "address": "$A$1",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 1,
  "current_region": "$A$1:$B$2",
  "formula": "=1+1.1",
  "formula_array": "=1+1,1",
  "height": 14.25,
  "last_cell": "$A$1",
  "left": 0.0,
  "name": "myname2",
  "number_format": "General",
  "row": 1,
  "row_height": 14.3,
  "shape": [
    1,
    1
  ],
  "size": 1,
  "top": 0.0,
  "value": 2.1,
  "width": 51.0
}
```

GET /books/<book_name_or_ix>/sheets

:

```
{
  "sheets": [
    {
      "charts": [
        "Chart 1"
      ],
      "name": "Sheet1",
      "names": [
        "Sheet1!myname1"
      ],
      "pictures": [
        "Picture 3"
      ],
      "shapes": [
        "Chart 1",
        "Picture 3"
      ],
      "used_range": "$A$1:$B$2"
    },
    {
      "charts": [],
      "name": "Sheet2",
      "names": [],
      "pictures": [],
      "shapes": [],
      "used_range": "$A$1"
    }
  ]
}
```

GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>

:

```
{
  "charts": [
    "Chart 1"
  ],
  "name": "Sheet1",
  "names": [
    "Sheet1!myname1"
  ],
  "pictures": [
    "Picture 3"
  ],
  "shapes": [
    "Chart 1",
```

( )

( )

```
    "Picture 3"
  ],
  "used_range": "$A$1:$B$2"
}
```

GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/charts

:

```
{
  "charts": [
    {
      "chart_type": "line",
      "height": 211.0,
      "left": 0.0,
      "name": "Chart 1",
      "top": 0.0,
      "width": 355.0
    }
  ]
}
```

GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/charts/<chart_name_or_ix>

:

```
{
  "chart_type": "line",
  "height": 211.0,
  "left": 0.0,
  "name": "Chart 1",
  "top": 0.0,
  "width": 355.0
}
```

GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/names

:

```
{
  "names": [
    {
      "name": "Sheet1!myname1",
      "refers_to": "=Sheet1!$B$2:$C$3"
    }
  ]
}
```

GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/names/<sheet_scope_name>

:

```
{
  "name": "Sheet1!myname1",
  "refers_to": "=Sheet1!$B$2:$C$3"
}
```

`GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/names/<sheet_scope_name>/range`

:

```
{
  "address": "$B$2:$C$3",
  "color": null,
  "column": 2,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "",
      ""
    ],
    [
      "",
      ""
    ]
  ],
  "formula_array": "",
  "height": 28.5,
  "last_cell": "$C$3",
  "left": 51.0,
  "name": "Sheet1!myname1",
  "number_format": "General",
  "row": 2,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 14.25,
  "value": [
    [
      null,
      null
    ],
    [
```

( )

( )

```
      null,
      null
    ]
  ],
  "width": 102.0
}
```

GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/pictures

:

```
{
  "pictures": [
    {
      "height": 100.0,
      "left": 0.0,
      "name": "Picture 3",
      "top": 0.0,
      "width": 100.0
    }
  ]
}
```

GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/pictures/<picture_name_or_ix>

:

```
{
  "height": 100.0,
  "left": 0.0,
  "name": "Picture 3",
  "top": 0.0,
  "width": 100.0
}
```

GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/range

:

```
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
```

( )

( )

```
      "=1+1.1",
      "a string"
    ],
    [
      "43395.0064583333",
      ""
    ]
  ],
  "formula_array": null,
  "height": 28.5,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
  "number_format": null,
  "row": 1,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
    [
      2.1,
      "a string"
    ],
    [
      "Mon, 22 Oct 2018 00:09:18 GMT",
      null
    ]
  ],
  "width": 102.0
}
```

`GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/range/<address>`

:

```
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
```

( )

( )

```
  "formula": [
    [
      "=1+1.1",
      "a string"
    ],
    [
      "43395.0064583333",
      ""
    ]
  ],
  "formula_array": null,
  "height": 28.5,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
  "number_format": null,
  "row": 1,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
    [
      2.1,
      "a string"
    ],
    [
      "Mon, 22 Oct 2018 00:09:18 GMT",
      null
    ]
  ],
  "width": 102.0
}
```

GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/shapes

:

```
{
  "shapes": [
    {
      "height": 211.0,
      "left": 0.0,
```

( )

( )

```
      "name": "Chart 1",
      "top": 0.0,
      "type": "chart",
      "width": 355.0
    },
    {
      "height": 100.0,
      "left": 0.0,
      "name": "Picture 3",
      "top": 0.0,
      "type": "picture",
      "width": 100.0
    }
  ]
}
```

GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/shapes/<shape_name_or_ix>

:

```
{
  "height": 211.0,
  "left": 0.0,
  "name": "Chart 1",
  "top": 0.0,
  "type": "chart",
  "width": 355.0
}
```

27.6.3 /apps

GET /apps

:

```
{
  "apps": [
    {
      "books": [
        "Book1",
        "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
        "Book4"
      ],
      "calculation": "automatic",
      "display_alerts": true,
      "pid": 1104,
```

( )

( )

```
      "screen_updating": true,
      "selection": "[Book1.xlsx]Sheet2!$A$1",
      "version": "16.0",
      "visible": true
    },
    {
      "books": [
        "Book2",
        "Book5"
      ],
      "calculation": "automatic",
      "display_alerts": true,
      "pid": 7920,
      "screen_updating": true,
      "selection": "[Book5]Sheet2!$A$1",
      "version": "16.0",
      "visible": true
    }
  ]
}
```

GET /apps/<pid>

:

```
{
  "books": [
    "Book1",
    "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
    "Book4"
  ],
  "calculation": "automatic",
  "display_alerts": true,
  "pid": 1104,
  "screen_updating": true,
  "selection": "[Book1.xlsx]Sheet2!$A$1",
  "version": "16.0",
  "visible": true
}
```

GET /apps/<pid>/books

:

```
{
  "books": [
    {
```

( )

(　)

```
      "app": 1104,
      "fullname": "Book1",
      "name": "Book1",
      "names": [],
      "selection": "Sheet2!$A$1",
      "sheets": [
        "Sheet1"
      ]
    },
    {
      "app": 1104,
      "fullname": "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
      "name": "Book1.xlsx",
      "names": [
        "Sheet1!myname1",
        "myname2"
      ],
      "selection": "Sheet2!$A$1",
      "sheets": [
        "Sheet1",
        "Sheet2"
      ]
    },
    {
      "app": 1104,
      "fullname": "Book4",
      "name": "Book4",
      "names": [],
      "selection": "Sheet2!$A$1",
      "sheets": [
        "Sheet1"
      ]
    }
  ]
}
```

`GET /apps/<pid>/books/<book_name_or_ix>`

:

```
{
  "app": 1104,
  "fullname": "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
  "name": "Book1.xlsx",
  "names": [
    "Sheet1!myname1",
```

(　)

( )

```
    "myname2"
  ],
  "selection": "Sheet2!$A$1",
  "sheets": [
    "Sheet1",
    "Sheet2"
  ]
}
```

`GET /apps/<pid>/books/<book_name_or_ix>/names`

:

```
{
  "names": [
    {
      "name": "Sheet1!myname1",
      "refers_to": "=Sheet1!$B$2:$C$3"
    },
    {
      "name": "myname2",
      "refers_to": "=Sheet1!$A$1"
    }
  ]
}
```

`GET /apps/<pid>/books/<book_name_or_ix>/names/<name>`

:

```
{
  "name": "myname2",
  "refers_to": "=Sheet1!$A$1"
}
```

`GET /apps/<pid>/books/<book_name_or_ix>/names/<name>/range`

:

```
{
  "address": "$A$1",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 1,
  "current_region": "$A$1:$B$2",
  "formula": "=1+1.1",
  "formula_array": "=1+1,1",
```

( )

( )

```
  "height": 14.25,
  "last_cell": "$A$1",
  "left": 0.0,
  "name": "myname2",
  "number_format": "General",
  "row": 1,
  "row_height": 14.3,
  "shape": [
    1,
    1
  ],
  "size": 1,
  "top": 0.0,
  "value": 2.1,
  "width": 51.0
}
```

`GET /apps/<pid>/books/<book_name_or_ix>/sheets`

:

```
{
  "sheets": [
    {
      "charts": [
        "Chart 1"
      ],
      "name": "Sheet1",
      "names": [
        "Sheet1!myname1"
      ],
      "pictures": [
        "Picture 3"
      ],
      "shapes": [
        "Chart 1",
        "Picture 3"
      ],
      "used_range": "$A$1:$B$2"
    },
    {
      "charts": [],
      "name": "Sheet2",
      "names": [],
      "pictures": [],
      "shapes": [],
```

( )

( )

```
      "used_range": "$A$1"
    }
  ]
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>

:

```
{
  "charts": [
    "Chart 1"
  ],
  "name": "Sheet1",
  "names": [
    "Sheet1!myname1"
  ],
  "pictures": [
    "Picture 3"
  ],
  "shapes": [
    "Chart 1",
    "Picture 3"
  ],
  "used_range": "$A$1:$B$2"
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/charts

:

```
{
  "charts": [
    {
      "chart_type": "line",
      "height": 211.0,
      "left": 0.0,
      "name": "Chart 1",
      "top": 0.0,
      "width": 355.0
    }
  ]
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/charts/<chart_name_or_ix>

:

```
{
  "chart_type": "line",
  "height": 211.0,
  "left": 0.0,
  "name": "Chart 1",
  "top": 0.0,
  "width": 355.0
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/names

:

```
{
  "names": [
    {
      "name": "Sheet1!myname1",
      "refers_to": "=Sheet1!$B$2:$C$3"
    }
  ]
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/names/<sheet_scope_name>

:

```
{
  "name": "Sheet1!myname1",
  "refers_to": "=Sheet1!$B$2:$C$3"
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/names/<sheet_scope_name>/ran

:

```
{
  "address": "$B$2:$C$3",
  "color": null,
  "column": 2,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "",
      ""
    ],
    [
```

( )

( )

```
      "",
      ""
    ]
  ],
  "formula_array": "",
  "height": 28.5,
  "last_cell": "$C$3",
  "left": 51.0,
  "name": "Sheet1!myname1",
  "number_format": "General",
  "row": 2,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 14.25,
  "value": [
    [
      null,
      null
    ],
    [
      null,
      null
    ]
  ],
  "width": 102.0
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/pictures

:

```
{
  "pictures": [
    {
      "height": 100.0,
      "left": 0.0,
      "name": "Picture 3",
      "top": 0.0,
      "width": 100.0
    }
  ]
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/pictures/<picture_name_or_ix

:

```
{
  "height": 100.0,
  "left": 0.0,
  "name": "Picture 3",
  "top": 0.0,
  "width": 100.0
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/range

:

```
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "=1+1.1",
      "a string"
    ],
    [
      "43395.0064583333",
      ""
    ]
  ],
  "formula_array": null,
  "height": 28.5,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
  "number_format": null,
  "row": 1,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
```

( )

( )

```
    [
      2.1,
      "a string"
    ],
    [
      "Mon, 22 Oct 2018 00:09:18 GMT",
      null
    ]
  ],
  "width": 102.0
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/range/<address>

:

```
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "=1+1.1",
      "a string"
    ],
    [
      "43395.0064583333",
      ""
    ]
  ],
  "formula_array": null,
  "height": 28.5,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
  "number_format": null,
  "row": 1,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
```

( )

( )

```
  "top": 0.0,
  "value": [
    [
      2.1,
      "a string"
    ],
    [
      "Mon, 22 Oct 2018 00:09:18 GMT",
      null
    ]
  ],
  "width": 102.0
}
```

`GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/shapes`

:

```
{
  "shapes": [
    {
      "height": 211.0,
      "left": 0.0,
      "name": "Chart 1",
      "top": 0.0,
      "type": "chart",
      "width": 355.0
    },
    {
      "height": 100.0,
      "left": 0.0,
      "name": "Picture 3",
      "top": 0.0,
      "type": "picture",
      "width": 100.0
    }
  ]
}
```

`GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/shapes/<shape_name_or_ix>`

:

```
{
  "height": 211.0,
  "left": 0.0,
  "name": "Chart 1",
```

( )

( )

```
  "top": 0.0,
  "type": "chart",
  "width": 355.0
}
```