pyxll

=python("in excel")

Integrating Excel and Python

Tony Roberts
tony@pyxll.com
https://www.pyxll.com

Introduction

Tony Roberts

tony@pyxll.com

Author of PyXLL (2010)

pyxll.com

and Jinx, the Excel Java Add-In (2018)

exceljava.com

Integrating Excel and Python

- **Is Excel still relevant?**
- Differences and Similarities with Jupyter Notebooks
- Reading and Writing Excel Files
- Automating Excel
- Writing Excel Add-Ins

Is Excel still relevant?

- **30 million** users… in 1996
- In 2019, best estimate is around **800 million** users
- Why? All *developers* know Excel sucks!
- Basic analysis and visualization can be done *without* programming
- Data flow is visual and intuitive

So why does Excel suck?

- VBA
  - Excel *power* users often over-use VBA to achieve complex results
  - VBA macros can sometimes make it harder to reason about behaviour
- VBA
  - Source version control is almost never applied to VBA code
  - Code *and data* usually embedded in sheets which get copied and emailed around
- Spaghetti Worksheets
  - A cell contains one value. Even small datasets need *lots* of cells!

What is Python good for?

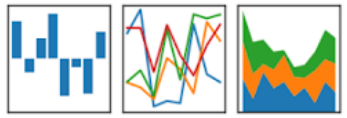
- Data Analysis and Visualization
- Machine Learning & AI
- Financial Modelling
- Scientific Computing
- Web Development
- Scripting / Automation

Examples of Python Packages

Scientific Computing

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



NumPy

SciPy

Data Access

Quandl

SQLAlchemy

AI & Machine Learning

scikit learn

theano

TensorFlow

K Keras

Jupyter notebooks FTW!

- Interactive tool for data analysis and visualization
- Access to full Python ecosystem
- Linear workflow
- Doesn't encourage code re-use
- Too technical for many end users

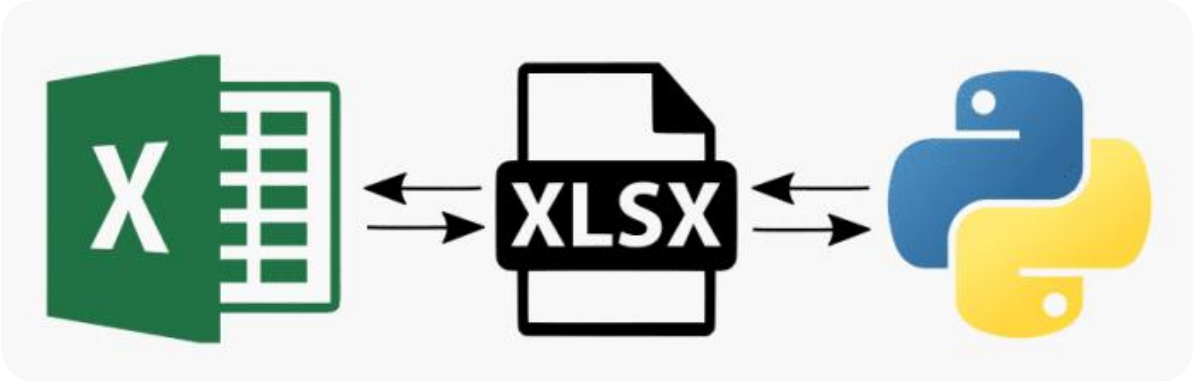
Can we have the best of both worlds?

- Keep Excel as a front-end **only**
- Use Python for *"heavy lifting"*
- Continue to develop Python code using familiar tools
- Stop storing code and data in Excel!

Integrating Excel and Python

- Is Excel still relevant?
- Differences and Similarities with Jupyter Notebooks
- **Reading and Writing Excel Files**
- Automating Excel
- Writing Excel Add-Ins

Reading and Writing Excel Files



Reading and Writing Excel Files

- openpyxl
  - Read and write xlsx files
  - Some content (e.g. charts) lost when reading and writing a file
- XlsxWriter
  - Write xlsx files
  - Faster and may use less memory than openpyxl
- xlrd / xlwt
  - Read and write old-style xls files
- xltable
  - Higher level abstraction for writing interactive reports
  - Uses XlsxWriter to output workbooks

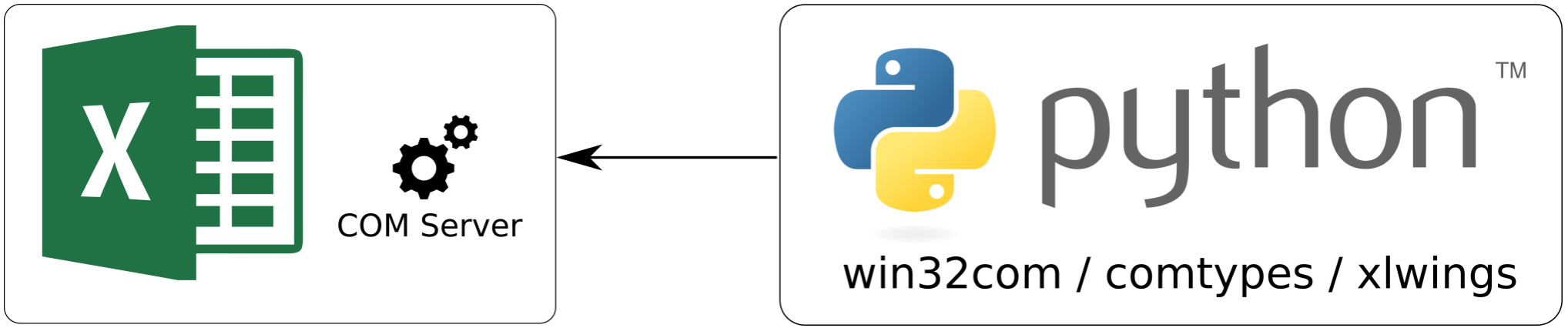
Example: openpyxl

```
from openpyxl import Workbook
wb = Workbook()
ws = wb.active  # Get the active worksheet
ws['A1'] = 42  # Data can be assigned directly to cells
ws.append([1, 2, 3])  # Rows can also be appended
ws['A2'] = datetime.datetime.now() # Python types converted
wb.save("sample.xlsx") # Save the file
```

Integrating Excel and Python

- What is Python and who uses it?
- Reading and Writing Excel Files
- **Automating Excel**
- Writing Excel Add-Ins

Excel Automation with Python



Excel Automation with Python

- win32com / pywin32
  - Excellent client and server side support for IDispatch based COM interfaces.
  - Requires additional C code to access non-dispatch based interfaces.
- comtypes
  - Pure python package capable of calling custom COM interfaces.
- xlwings
  - Wrapper around win32com for Excel on Windows and appscript on Mac.
  - Support for calling Python from VBA.

Example: win32com

- Connect to Excel from Python
- Modify active worksheet
- Connect to events

Example: xlwings

- Wrapper around win32com
- Write pandas DataFrames to Excel
- VBA functions for calling Python server

Integrating Excel and Python

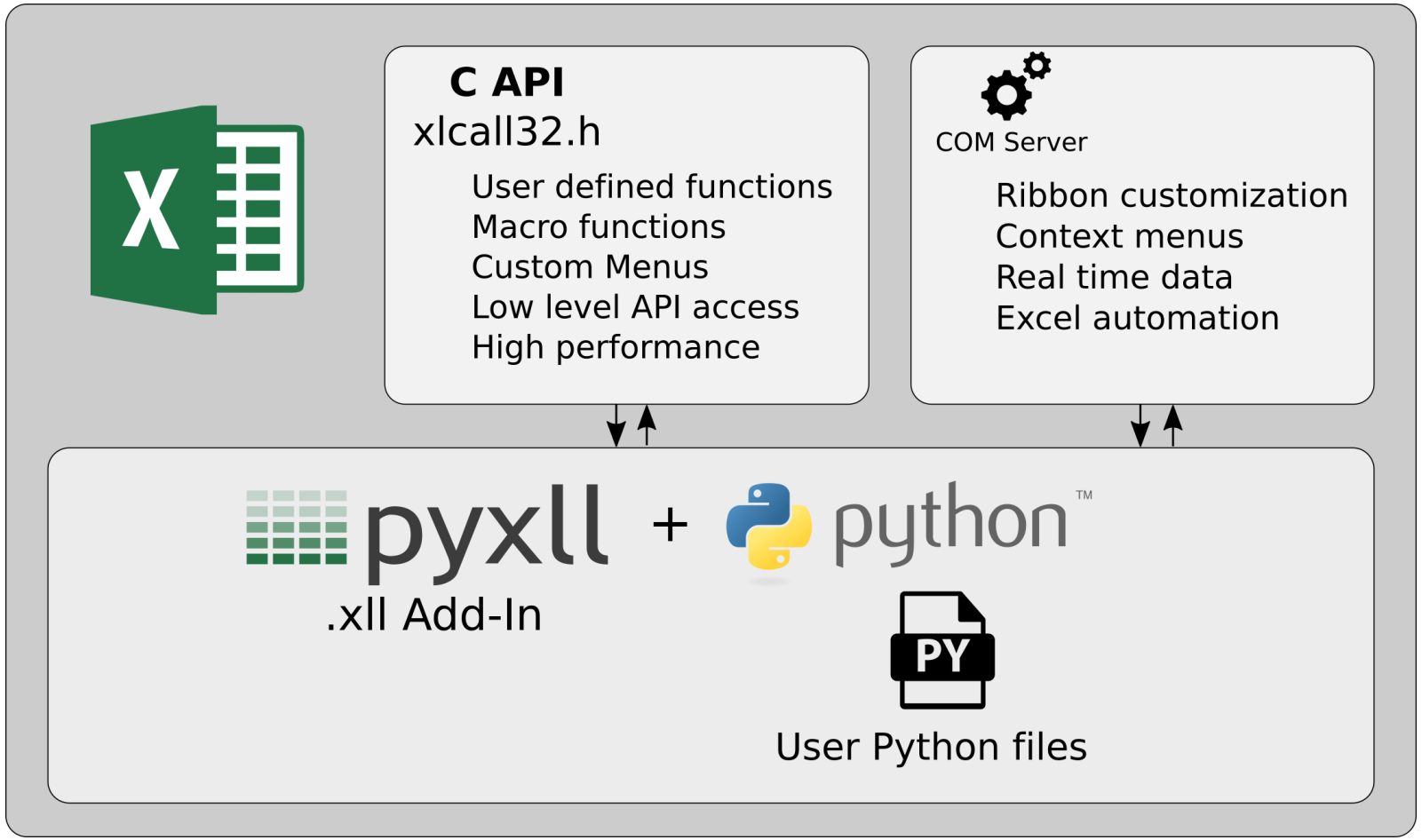
- What is Python and who uses it?
- Reading and Writing Excel Files
- Automating Excel
- **Writing Excel Add-Ins**

Writing Excel Add-Ins in Python

Expose Python functions to Excel as

- User defined functions (UDFs)
  - Volatile
  - Multi-threaded
  - Asynchronous
  - Real Time Data
  - Macro Equivalent
  - Dynamic array functions
- Macro functions
  - Call back into Excel
- Ribbon Controls
- Toolbar and Context Menu Items

Writing Excel Add-Ins in Python



Demo: User Defined Functions

- Decorate functions using @xl_func
- Add optional function signature for type information
- Array functions can be automatically resized
- Objects returned as object handles
- Return small DataFrames as arrays
- Return large DataFrames as objects

Standard Types

- int, float, bool, str
- 1d and 2d arrays, e.g. "float[]" and "float[][]"
- NumPy arrays, e.g. "numpy_array<float, ndim=2>"
- Pandas DataFrame and Series, e.g. "dataframe<index=False>"
- Python Objects
- "var" type that accepts any type
- "xl_cell" for cell info as well as value

Custom Types

Add your own type converters using

- @xl_arg_type
- @xl_return_type

Access all type converters using

- pyxll.get_type_converter

Demo: Customizing the Ribbon

- Write/modify a ribbon xml file
  - Add tabs / groups / controls to <ribbon> element
  - Add controls / sub-menus to <contextMenus> element
  - Actions are bound to Python functions
- Configure pyxll.cfg
- Can be done programmatically using pyxll API

Demo: Python embedded in Excel

Demo of Jupyter / IPython kernel running inside Excel.

Demo: Real Time Data

- RTD functions use return type "rtd<T>"
- Values returned to Excel using *RTD.value*
- Use background thread or coroutine
- Return value can be any type
- Errors returned using *RTD.set_error*

Demo: Machine Learning

- Models can be developed outside of Excel
- Trained models can be deployed to Excel for business users
- Same model used for automatic processing and user interaction

Questions

Download materials from

http://github.com/pyxll/pylondon-2019

Contact me

tony@pyxll.com / @pyxll

Also available for Java / Scala / Kotlin

https://exceljava.com