Python for MATLAB Developent

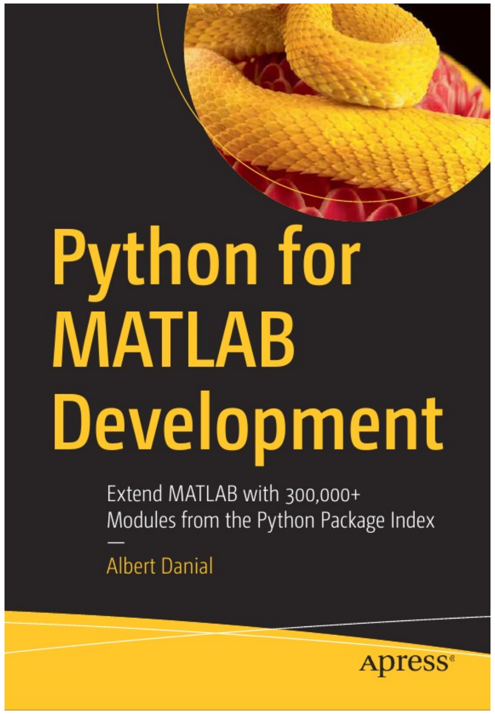
Al Danial

Introduction

- MATLAB's py module provides a direct interface to Python!
- Primary aspects of the MATLAB→Python interface are
  - Configuring MATLAB to recognize your Python installation
  - Updating MATLAB's search path to find your Python code
  - Importing Python modules
  - Passing MATLAB variables to Python functions
  - Converting Python return values to MATLAB variables
  - Writing Python bridge functions to cover interface gaps
- Each aspect is demonstrated with examples

About me

- Aerospace engineer (BAE from Georgia Tech, MSAA & Ph.D. from Purdue); currently Senior Staff Engineer at Northrop Grumman (25 years), former NASTRAN developer at MSC.Software (3 years)
- Software developer in MATLAB (since v4 in 1990), Python (since v2.6 in 2006), C++, Fortran, Perl (github.com/AlDanial/cloc)
- Author of *Python for MATLAB Development* (Apress 2022)


Python for MATLAB Development
Extend MATLAB with 300,000+ Modules from the Python Package Index
Albert Danial
Apress

To get the most from this talk:

- Computer with
  - MATLAB version 2020b or newer (ideal: 2022a)
  - Anaconda Python version 2018 or newer (ideal: 2021.12)
- Familiarity with Python

To start: Tell MATLAB Where To Find Your Python Installation

```
>> pyenv('Version', ...
         '/usr/local/anaconda3/2020.07/envs/matpy/bin/python');

                      - or -

>> pyenv('Version', '3.9');
```

- A good location for the pyenv command is `startup.m`

```
>> edit(fullfile(userpath,'startup.m'))
```

Example 1: How much memory is in use?

```
>> memory
Error using memory
Function MEMORY is not available on this platform.
```

- MATLAB's `memory` only works on Windows (above was Linux)
- Python's `psutil` module is cross-platform:

an ipython session

```
In : import psutil as ps
In : ps.virtual_memory()
Out: svmem(total=8236642304, available=3313963008, percent=59.8, used=3974819840,
free=294387712, active=2696413184, inactive=4374761472, buffers=481042432,
cached=3486392320, shared=655716352, slab=464904192)
```

Get memory in MATLAB via Python

```
>> ps = py.importlib.import_module('psutil');

>> m = ps.virtual_memory()
 Python svmem with properties:

       [lines deleted]

    svmem(total=8236642304, available=2251964416,
percent=72.7, used=4993241088, free=154984448,
active=2384990208, inactive=4833411072,
buffers=292765696, cached=2795651072, shared=691990528,
slab=411570176)

>> m.used/m.total
ans =
    0.6148
```

MATLAB equivalent of `import psutil as ps`

`m.used` and `m.total` are Python variables; math operations with them work fine

Aside: Simplify Module Imports

- `X = py.importlib.import_module('X')` is a lot to type!
- I use two shortcuts:

  1) a function handle, for use in .m files

  ```
  Im = @py.importlib.import_module
  ```

  2) a wrapper function, for interactive use

  ```
  function [module] = imp(module_name)
      module = py.importlib.import_module(module_name);
  end
  ```

- My `startup.m` defines `Im`, and `imp.m` is in my MATLAB `path`.

Example 2: calendar and MATLAB v. Python input arguments

- Python can print a monthly calendar (like the `cal` command on Linux and macOS):

```
In : import calendar
In : calendar.prmonth(2022, 5)
      May 2022
Mo Tu We Th Fr Sa Su
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

Calendar in MATLAB (attempt #1)

```
>> py.calendar.prmonth(2022, 5)
Error using calendar>__getitem__ (line 59)
Python Error: TypeError: list indices must be
integers or slices, not float

Error in calendar>formatmonthname (line 341)

Error in calendar>formatmonth (line 358)

Error in calendar>prmonth (line 350)
```

- MATLAB numeric literals are *`doubles`*

```
>> class(2022)
    'double'
```

- `calendar.prmonth()` expects *integers*

Calendar in MATLAB (attempt #2)

```
>> py.calendar.prmonth(int64(2022),int64(5))
      May 2022
Mo Tu We Th Fr Sa Su
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

- `int64()` appears frequently in MATLAB→Python calls

Python Variables Retain Access to Their Member Functions

```
>> np = imp('numpy');
>> x = np.random.rand(int64(2),int64(3))
x =
  Python ndarray:
   3.2646e-01   6.5279e-01   2.1619e-01
   1.8322e-01   2.9653e-01   1.8462e-01

>> [x.min(), x.max(), x.std()]
   1.8322e-01   6.5279e-01   1.6256e-01
```

- Tab expansion on “x.” in the IDE shows x’s methods
- Indexing Python objects can be a challenge

```
>> x(1,2)
Array formation and parentheses-style indexing with objects of
class 'py.numpy.ndarray' is not allowed.  Use
objects of class 'py.numpy.ndarray' only as scalars or use a cell
array.
```

- Possible solutions:
  - Convert x to a MATLAB variable
  - Write a bridge module with access function

Example 3: Call your own Python code

- A simple Python function in file `txy.py`:

```
# txy.py
import numpy as np
from datetime import datetime
def F():
    return { 't' : datetime.now(),
             'x' : np.arange(12).reshape(2,6),
             'y' : ['a list', 'with strings']}
```

- Lives in directory `/home/al/project7`, so add this to the Python search path in MATLAB

same concept as `addpath` *in MATLAB*

```
>> sys_path = py.sys.path
>> sys_path.append('/home/al/project7')
```

Call our Own Python Function

- Call `txy.F()` in MATLAB:

```
>> txy = imp('txy');
>> z = txy.F()
z =
  Python dict with no properties.
    {'t': datetime.datetime(2022, 3, 13, 13, 55, 5, 801403),
     'x': array([[ 0,  1,  2,  3,  4,  5], [ 6,  7,  8,  9, 10, 11]]),
     'y': ['a list', 'with strings']}
```

- ...but Python functions return Python variables.
- Takes extra effort to get values of interest.

```
>> double(z.get('x'))
     0     1     2     3     4     5
     6     7     8     9    10    11
```

- Most of the time, would rather have MATLAB variables

`py2mat.m`: Python to MATLAB Data Converter

- Given a Python variable, `py2mat()` returns its values as a MATLAB variable.

```
>> m = py2mat(z)
  struct with fields:
    t: 26-Feb-2022 18:30:30
    x: [2×6 int64]
    y: {["a list"]  ["with strings"]}

>> m.x
  2×6 int64 matrix
    0    1    2    3    4    5
    6    7    8    9   10   11
```

idiomatic MATLAB; more intuitive than `double(z.get('x'))`

https://github.com/Apress/python-for-matlab-development/blob/main/code/matlab_py/py2mat.m

- `py2mat()` supports real and complex NumPy arrays (preserves type and bit size), dates with timezone, lists, dicts, strings, tuples, sets, SciPy sparse matrices

Example 4: Read a YAML file

- YAML is convenient format for storing program configuration data; is much less tedious than XML.
- Sample file:

```yaml
# optim_config.yaml
max_iter : 1000
newmark :
    alpha : 0.25
    beta : 0.5
input_dir : "/xfer/sim_data/2022/05/17"
tune_coeff : [1.2e-4, -3.25, 58.2]
```

MATLAB Doesn’t Know YAML

```
>> config = load('optim_config.yaml')
Error using load
Unable to read file 'optim_config.yaml'. Input
must be a MAT-file or an ASCII file containing
numeric data with same number of columns in each
row.
```

- MATLAB solutions exist on Github and the FileExchange
- Alternatively, use Python…

Read YAML with Python

```
>> yaml = imp('yaml');

>> config = py2mat(yaml.safe_load(py.open('optim_config.yaml')))
config =
  struct with fields:
      max_iter: 1000
       newmark: [1×1 struct]
     input_dir: "/xfer/sim_data/2022/05/17"
    tune_coeff: {[1.2000e-04]  [-3.2500]  [58.2000]}

>> config.newmark
  struct with fields:
    alpha: 0.2500
     beta: 0.5000
```

```
# optim_config.yaml
max_iter : 1000
newmark :
    alpha : 0.25
    beta : 0.5
input_dir : "/xfer/sim_data/2022/05/17"
tune_coeff : [1.2e-4, -3.25, 58.2]
```

py2mat.m Inverse: mat2py.m

- Python functions require Python arguments
- MATLAB automatically maps simple MATLAB variables to Python variables when calling Python functions.
- More complex data? Use `mat2py.m`
  https://github.com/Apress/python-for-matlab-development/blob/main/code/matlab_py/mat2py.m
- Example of a complex MATLAB variable:

```
soln.converged = true;
soln.error = 5.98435e-4;
soln.shape = { 8.8 -3.1 };
soln.v(1).Ax = 4;
soln.v(1).Bx = [.5 .5 .5];
soln.v(2).Ax = -3;
soln.v(2).Bx = [.45 -.35 2.5];
```

`mat2py.m` Example

```
>> mat2py(soln)
  Python dict with no properties.

{'converged': True,
 'error': 0.000598435,
 'shape': [8.8, -3.1],
 'v': [{'Ax': 4.0,
        'Bx': array([0.5, 0.5, 0.5])},
       {'Ax': -3.0,
        'Bx': array([ 0.45, -0.35,  2.5 ])}]}
```

- `mat2py.m` converts dense and sparse matrices, cell arrays, structs, strings, datetimes.

Example 5: Bridge Modules

- We saw earlier that a NumPy array cannot be indexed in MATLAB:

```
>> np = imp('numpy');
>> x = np.arange(12).reshape(int64(2),int64(6))
x =
  Python ndarray:
     0     1     2     3     4     5
     6     7     8     9    10    11

>> x(2,4)
Array formation and parentheses-style indexing with objects of class
'py.numpy.ndarray' is not allowed.  Use objects of class 'py.numpy.ndarray'
only as scalars or use a cell array.
```

- Write a *bridge module* to provide missing functionality:

```
# bridge_numpy_index.py
def ind(z, row, col):
    return z[int(row)-1,int(col)-1]
```

subtracting 1 lets us use one-based indexing in MATLAB

cast to integer lets us avoid `int64()` in MATLAB

Bridge Module to Index NumPy Array

- Import the bridge and use its function(s) to get access:

```
>> br = imp('bridge_numpy_index');
>> x = np.arange(12).reshape(int64(2),int64(6))
x =
  Python ndarray:
     0     1     2     3     4     5
     6     7     8     9    10    11

>> br.ind(x,2,4)
ans =

     9
```

MATLAB one-based indexing

- A smarter, *n*-dimensional version:

```
# bridge_numpy_index.py
def ind(z, *i):
    int_minus_1 = tuple([int(_) - 1 for _ in I])
    return z[int_minus_1]
```

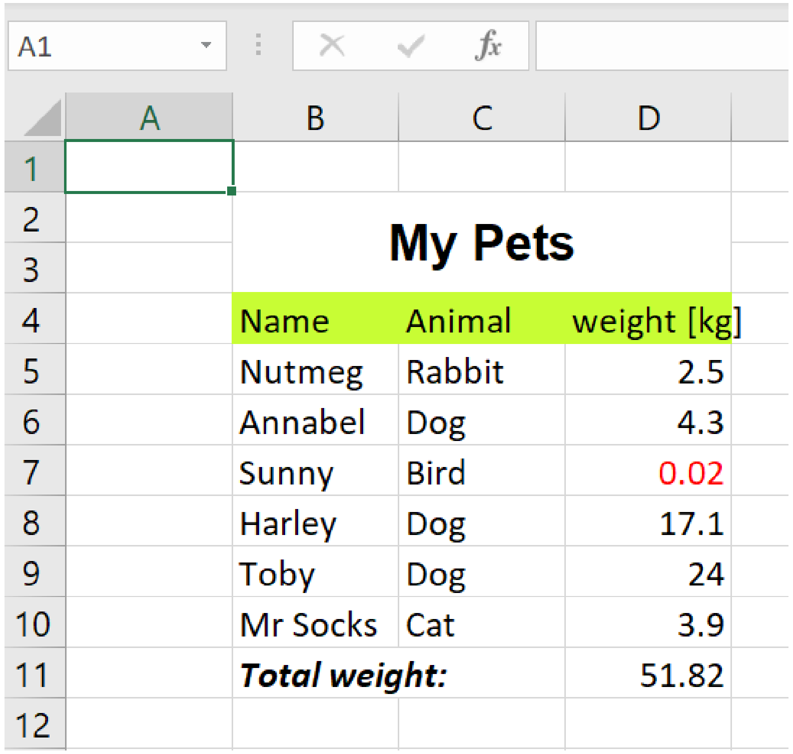
- Want index slices, submatrices? Write more functions.

Process Review

1. Tell MATLAB where Python is installed
```
>> pyenv('Version', '/path/to/python');
```
2. Expand `sys.path` so Python can find your code
```
>> sys_path = py.sys.path
>> sys_path.append('/path/to/your/code');
```
3. Import Python modules
```
>> alias = py.importlib.import_module('module_name');
```
4. Call Python functions with Python arguments
```
>> x = alias.Fn(int64(n), mat2py(matlab_variable));
```
5. Convert Python return values to MATLAB variables
```
>> m = py2mat(x);
```
6. Write a bridge module to span interface gaps.
```
>> bridge = py.importlib.import_module('bridge_numpy_index');
>> m = bridge.ind(x,1,2);
```

- Ready for a bigger challenge!

Example 6: Write Formatted Excel `.xlsx`



	A	B	C	D
1				
2		My Pets		
3				
4		Name	Animal	weight [kg]
5		Nutmeg	Rabbit	2.5
6		Annabel	Dog	4.3
7		Sunny	Bird	0.02
8		Harley	Dog	17.1
9		Toby	Dog	24
10		Mr Socks	Cat	3.9
11		***Total weight:***		51.82
12				

- Custom fonts, size, color
- Merged cells
- Background colors
- Equations

- Can do it in MATLAB on Windows with COM
- Can do it in MATLAB on Linux, macOS, Windows without COM using Python `openpyxl` module

Not a One-Liner!

- Best approach for an involved MATLAB-calling-Python solution:
  - Write a working prototype entirely in Python
  - Implement each Python line in MATLAB

- Reference Python solution will be useful for troubleshooting in MATLAB

demo_openpyxl.m, demo_openpyxl.py (1/12)

```
% Al Danial, David Garrison
Im = @py.importlib.import_module;
OP = Im("openpyxl");
styles = Im("openpyxl.styles");
Font = styles.Font;
Alignment = styles.Alignment;
PatternFill = styles.PatternFill;
book = OP.Workbook();
sheet = book.active;
sheet.title = "Pets by weight";

% font styles, background color
ft_title = Font(...
    pyargs("name","Arial", ...
    "size",int64(14),"bold",py.True));
ft_red = Font(color="00FF0000");
ft_italics = Font(bold=py.True,...
                  italic=py.True);
bg_green = PatternFill( ...
    fgColor="C5FD2F", fill_type="solid");

sheet.merge_cells("B2:D3");
B2 = sheet.cell(2,2);
B2.value = "My Pets";
B2.font = ft_title;
B2.alignment = Alignment(...
 horizontal="center", vertical="center");
```

```
#!/usr/bin/env python3

import openpyxl as OP
import openpyxl.styles as styles
Font = styles.Font
Alignment = styles.Alignment
PatternFill = styles.PatternFill
book = OP.Workbook()
sheet = book.active
sheet.title = "Pets by weight"

# font styles, background color
ft_title = Font(name="Arial",
                size=14,
                bold=True)
ft_red = Font(color="00FF0000")
ft_italics = Font(bold=True,
                  italic=True)
bg_green = PatternFill(
    fgColor="C5FD2F",fill_type="solid")

sheet.merge_cells("B2:D3")
B2 = sheet.cell(2,2)
B2.value = "My Pets"
B2.font = ft_title
B2.alignment = Alignment(
  horizontal="center",vertical="center")
```

demo_openpyxl.m, demo_openpyxl.py (2/12)

```
% Al Danial, David Garrison
Im = @py.importlib.import_module;
OP = Im("openpyxl");
styles = Im("openpyxl.styles");
Font = styles.Font;
Alignment = styles.Alignment;
PatternFill = styles.PatternFill;
book = OP.Workbook();
sheet = book.active;
sheet.title = "Pets by weight";

% font styles, background color
ft_title = Font(...
    pyargs("name","Arial", ...
    "size",int64(14),"bold",py.True));
ft_red = Font(color="00FF0000");
ft_italics = Font(bold=py.True,...
                  italic=py.True);
bg_green = PatternFill( ...
    fgColor="C5FD2F", fill_type="solid");

sheet.merge_cells("B2:D3");
B2 = sheet.cell(2,2);
B2.value = "My Pets";
B2.font = ft_title;
B2.alignment = Alignment(...
 horizontal="center", vertical="center");
```

```
#!/usr/bin/env python3

import openpyxl as OP
import openpyxl.styles as styles
Font = styles.Font
Alignment = styles.Alignment
PatternFill = styles.PatternFill
book = OP.Workbook()
sheet = book.active
sheet.title = "Pets by weight"

# font styles, background color
ft_title = Font(name="Arial",
                size=14,
                bold=True)
ft_red = Font(color="00FF0000")
ft_italics = Font(bold=True,
                  italic=True)
bg_green = PatternFill(
    fgColor="C5FD2F",fill_type="solid")

sheet.merge_cells("B2:D3")
B2 = sheet.cell(2,2)
B2.value = "My Pets"
B2.font = ft_title
B2.alignment = Alignment(
  horizontal="center",vertical="center")
```

demo_openpyxl.m, demo_openpyxl.py (3/12)

```matlab
% Al Danial, David Garrison
Im = @py.importlib.import_module;
OP = Im("openpyxl");
styles = Im("openpyxl.styles");
Font = styles.Font;
Alignment = styles.Alignment;
PatternFill = styles.PatternFill;
book = OP.Workbook();
sheet = book.active;
sheet.title = "Pets by weight";

% font styles, background color
ft_title = Font(...
    pyargs("name","Arial", ...
    "size",int64(14),"bold",py.True));
ft_red = Font(color="00FF0000");
ft_italics = Font(bold=py.True,...
                  italic=py.True);
bg_green = PatternFill( ...
    fgColor="C5FD2F", fill_type="solid");

sheet.merge_cells("B2:D3");
B2 = sheet.cell(2,2);
B2.value = "My Pets";
B2.font = ft_title;
B2.alignment = Alignment(...
 horizontal="center", vertical="center");
```

```python
#!/usr/bin/env python3

import openpyxl as OP
import openpyxl.styles as styles
Font = styles.Font
Alignment = styles.Alignment
PatternFill = styles.PatternFill
book = OP.Workbook()
sheet = book.active
sheet.title = "Pets by weight"

# font styles, background color
ft_title = Font(name="Arial",
                size=14,
                bold=True)
ft_red = Font(color="00FF0000")
ft_italics = Font(bold=True,
                  italic=True)
bg_green = PatternFill(
    fgColor="C5FD2F",fill_type="solid")

sheet.merge_cells("B2:D3")
B2 = sheet.cell(2,2)
B2.value = "My Pets"
B2.font = ft_title
B2.alignment = Alignment(
  horizontal="center",vertical="center")
```

demo_openpyxl.m, demo_openpyxl.py (4/12)

```
% Al Danial, David Garrison
Im = @py.importlib.import_module;
OP = Im("openpyxl");
styles = Im("openpyxl.styles");
Font = styles.Font;
Alignment = styles.Alignment;
PatternFill = styles.PatternFill;
book = OP.Workbook();
sheet = book.active;
sheet.title = "Pets by weight";

% font styles, background color
ft_title = Font(...
    pyargs("name","Arial", ...
    "size",int64(14),"bold",py.True));
ft_red = Font(color="00FF0000");
ft_italics = Font(bold=py.True,...
                  italic=py.True);
bg_green = PatternFill( ...
    fgColor="C5FD2F", fill_type="solid");

sheet.merge_cells("B2:D3");
B2 = sheet.cell(2,2);
B2.value = "My Pets";
B2.font = ft_title;
B2.alignment = Alignment(...
 horizontal="center", vertical="center");
```

```
#!/usr/bin/env python3

import openpyxl as OP
import openpyxl.styles as styles
Font = styles.Font
Alignment = styles.Alignment
PatternFill = styles.PatternFill
book = OP.Workbook()
sheet = book.active
sheet.title = "Pets by weight"

# font styles, background color
ft_title = Font(name="Arial",
                size=14,
                bold=True)
ft_red = Font(color="00FF0000")
ft_italics = Font(bold=True,
                  italic=True)
bg_green = PatternFill(
    fgColor="C5FD2F",fill_type="solid")

sheet.merge_cells("B2:D3")
B2 = sheet.cell(2,2)
B2.value = "My Pets"
B2.font = ft_title
B2.alignment = Alignment(
  horizontal="center",vertical="center")
```

demo_openpyxl.m, demo_openpyxl.py (5/12)

```
% Al Danial, David Garrison
Im = @py.importlib.import_module;
OP = Im("openpyxl");
styles = Im("openpyxl.styles");
Font = styles.Font;
Alignment = styles.Alignment;
PatternFill = styles.PatternFill;
book = OP.Workbook();
sheet = book.active;
sheet.title = "Pets by weight";

% font styles, background color
ft_title = Font(...
    pyargs("name","Arial", ...
    "size",int64(14),"bold",py.True));
ft_red = Font(color="00FF0000");
ft_italics = Font(bold=py.True,...
                  italic=py.True);
bg_green = PatternFill( ...
    fgColor="C5FD2F", fill_type="solid");

sheet.merge_cells("B2:D3");
B2 = sheet.cell(2,2);
B2.value = "My Pets";
B2.font = ft_title;
B2.alignment = Alignment(...
 horizontal="center", vertical="center");
```

before 2022a, need pyargs('x',y)

2022a allows x=y!

```
#!/usr/bin/env python3

import openpyxl as OP
import openpyxl.styles as styles
Font = styles.Font
Alignment = styles.Alignment
PatternFill = styles.PatternFill
book = OP.Workbook()
sheet = book.active
sheet.title = "Pets by weight"

# font styles, background color
ft_title = Font(name="Arial",
                size=14,
                bold=True)
ft_red = Font(color="00FF0000")
ft_italics = Font(bold=True,
                  italic=True)
bg_green = PatternFill(
    fgColor="C5FD2F",fill_type="solid")

sheet.merge_cells("B2:D3")
B2 = sheet.cell(2,2)
B2.value = "My Pets"
B2.font = ft_title
B2.alignment = Alignment(
  horizontal="center",vertical="center")
```

demo_openpyxl.m, demo_openpyxl.py (6/12)

```matlab
% Al Danial, David Garrison
Im = @py.importlib.import_module;
OP = Im("openpyxl");
styles = Im("openpyxl.styles");
Font = styles.Font;
Alignment = styles.Alignment;
PatternFill = styles.PatternFill;
book = OP.Workbook();
sheet = book.active;
sheet.title = "Pets by weight";

% font styles, background color
ft_title = Font(...
    pyargs("name","Arial", ...
    "size",int64(14),"bold",py.True));
ft_red = Font(color="00FF0000");
ft_italics = Font(bold=py.True,...
                  italic=py.True);
bg_green = PatternFill( ...
    fgColor="C5FD2F", fill_type="solid");

sheet.merge_cells("B2:D3");
B2 = sheet.cell(2,2);
B2.value = "My Pets";
B2.font = ft_title;
B2.alignment = Alignment(...
 horizontal="center", vertical="center");
```

```python
#!/usr/bin/env python3

import openpyxl as OP
import openpyxl.styles as styles
Font = styles.Font
Alignment = styles.Alignment
PatternFill = styles.PatternFill
book = OP.Workbook()
sheet = book.active
sheet.title = "Pets by weight"

# font styles, background color
ft_title = Font(name="Arial",
                size=14,
                bold=True)
ft_red = Font(color="00FF0000")
ft_italics = Font(bold=True,
                  italic=True)
bg_green = PatternFill(
    fgColor="C5FD2F",fill_type="solid")

sheet.merge_cells("B2:D3")
B2 = sheet.cell(2,2)
B2.value = "My Pets"
B2.font = ft_title
B2.alignment = Alignment(
  horizontal="center",vertical="center")
```

demo_openpyxl.m, demo_openpyxl.py (7/12)

```
% column headings
category={"Name","Animal","weight [kg]"};
row = int64(4); col = int64(1);
for i = 1:length(category)
  nextCell = sheet.cell(row, col+i);
  nextCell.value = category{i};
  nextCell.fill = bg_green;
end

pets = {{"Nutmeg", "Rabbit", 2.5},...
        {"Annabel", "Dog", 4.3},  ...
        {"Sunny", "Bird", 0.02},  ...
        {"Harley", "Dog", 17.1},  ...
        {"Toby", "Dog", 24.0},    ...
        {"Mr Socks", "Cat", 3.9}};

for P = pets
  row = row + 1;
  for j = 1:length(category)
    nextCell = cell(sheet,row,col+j);
    nextCell.value = P{1}{j};
    if j == 3 && P{1}{j} < 0.1
      nextCell = cell(sheet,row,col+j);
      nextCell.font = ft_red;
    end
  end
end
```

```
# column headings
category=["Name","Animal","weight [kg"]
row, col = 4, 2
for i in range(len(category)):
  nextCell = sheet.cell(row, col+i)
  nextCell.value = category[i]
  nextCell.fill = bg_green

pets = [["Nutmeg", "Rabbit", 2.5],
        ["Annabel", "Dog", 4.3],
        ["Sunny", "Bird", 0.02],
        ["Harley", "Dog", 17.1],
        ["Toby", "Dog", 24.0],
        ["Mr Socks", "Cat", 3.9]]

for P in pets:
  row += 1
  for j in range(len(category)):
    cell = sheet.cell(row,col+j,
                      P[j])
    if j == 2 and P[j] < 0.1:
      nextCell = sheet.cell(row,col+j)
      nextCell.font = ft_red
```

demo_openpyxl.m, demo_openpyxl.py (8/12)

```
% column headings
category={"Name","Animal","weight [kg]"};
row = int64(4); col = int64(1);
for i = 1:length(category)
  nextCell = sheet.cell(row, col+i);
  nextCell.value = category{i};
  nextCell.fill = bg_green;
end

pets = {{"Nutmeg", "Rabbit", 2.5},...
        {"Annabel", "Dog", 4.3},  ...
        {"Sunny", "Bird", 0.02},  ...
        {"Harley", "Dog", 17.1},  ...
        {"Toby", "Dog", 24.0},    ...
        {"Mr Socks", "Cat", 3.9}};

for P = pets
  row = row + 1;
  for j = 1:length(category)
    nextCell = sheet(cell,row,col+j);
    nextCell.value = P{1}{j};
    if j == 3 && P{1}{j} < 0.1
      nextCell = sheet(cell,row,col+j);
      nextCell.font = ft_red;
    end
  end
end
```

```
# column headings
category=["Name","Animal","weight [kg]"]
row, col = 4, 2
for i in range(len(category)):
  nextCell = sheet.cell(row, col+i)
  nextCell.value = category[i]
  nextCell.fill = bg_green

pets = [["Nutmeg", "Rabbit", 2.5],
        ["Annabel", "Dog", 4.3],
        ["Sunny", "Bird", 0.02],
        ["Harley", "Dog", 17.1],
        ["Toby", "Dog", 24.0],
        ["Mr Socks", "Cat", 3.9]]

for P in pets:
  row += 1
  for j in range(len(category)):
    cell = sheet.cell(row,col+j,
                      P[j])
    if j == 2 and P[j] < 0.1:
      nextCell = sheet.cell(row,col+j)
      nextCell.font = ft_red
```

demo_openpyxl.m, demo_openpyxl.py (9/12)

```matlab
% column headings
category={"Name","Animal","weight [kg]"};
row = int64(4); col = int64(1);
for i = 1:length(category)
  nextCell = sheet.cell(row, col+i);
  nextCell.value = category{i};
  nextCell.fill = bg_green;
end

pets = {{"Nutmeg", "Rabbit", 2.5},...
        {"Annabel", "Dog", 4.3},  ...
        {"Sunny", "Bird", 0.02},  ...
        {"Harley", "Dog", 17.1},  ...
        {"Toby", "Dog", 24.0},    ...
        {"Mr Socks", "Cat", 3.9}};

for P = pets
  row = row + 1;
  for j = 1:length(category)
    nextCell = sheet.cell,row,col+j);
    nextCell.value = P{1}{j};
    if j == 3 && P{1}{j} < 0.1
      nextCell = sheet.cell,row,col+j);
      nextCell.font = ft_red;
    end
  end
end
```

```python
# column headings
category=["Name","Animal","weight [kg]"]
row, col = 4, 2
for i in range(len(category)):
  nextCell = sheet.cell(row, col+i)
  nextCell.value = category[i]
  nextCell.fill = bg_green

pets = [["Nutmeg", "Rabbit", 2.5],
        ["Annabel", "Dog", 4.3],
        ["Sunny", "Bird", 0.02],
        ["Harley", "Dog", 17.1],
        ["Toby", "Dog", 24.0],
        ["Mr Socks", "Cat", 3.9]]

for P in pets:
  row += 1
  for j in range(len(category)):
    cell = sheet.cell(row,col+j,
                      P[j])
    if j == 2 and P[j] < 0.1:
      nextCell = sheet.cell(row,col+j)
      nextCell.font = ft_red
```

demo_openpyxl.m, demo_openpyxl.py (10/12)

```
% equation to sum all weights
eqn = sprintf("=SUM(D4:D%d)", row);
nextCell = sheet.cell(row+1, 4);
nextCell.value = eqn;

nextCell = sheet.cell(row+1, 2);
nextCell.value = "Total weight:";
nextCell.font = ft_italics;

book.save("pets.xlsx")
```

```
# equation to sum all weights
eqn = f"=SUM(D4:{row+1})"
nextCell = sheet.cell(row+1, 4)
nextCell.value = eqn

nextCell = sheet.cell(row+1, 2)
nextCell.value = "Total weight:"
nextCell.font = ft_italics;

book.save("pets.xlsx")
```

demo_openpyxl.m, demo_openpyxl.py (11/12)

```
% equation to sum all weights
eqn = sprintf("=SUM(D4:D%d)", row);
nextCell = sheet.cell(row+1, 4);
nextCell.value = eqn;

nextCell = sheet.cell(row+1, 2);
nextCell.value = "Total weight:";
nextCell.font = ft_italics;

book.save("pets.xlsx")
```

```
# equation to sum all weights
eqn = f"=SUM(D4:{row+1})"
nextCell = sheet.cell(row+1, 4)
nextCell.value = eqn

nextCell = sheet.cell(row+1, 2)
nextCell.value = "Total weight:"
nextCell.font = ft_italics;

book.save("pets.xlsx")
```

demo_openpyxl.m, demo_openpyxl.py (12/12)

```
% equation to sum all weights
eqn = sprintf("=SUM(D4:D%d)", row);
nextCell = sheet.cell(row+1, 4);
nextCell.value = eqn;

nextCell = sheet.cell(row+1, 2);
nextCell.value = "Total weight:";
nextCell.font = ft_italics;

book.save("pets.xlsx")
```

```
# equation to sum all weights
eqn = f"=SUM(D4:{row+1})"
nextCell = sheet.cell(row+1, 4)
nextCell.value = eqn

nextCell = sheet.cell(row+1, 2)
nextCell.value = "Total weight:"
nextCell.font = ft_italics;

book.save("pets.xlsx")
```

MATLAB/Python Code Challenges

- Dual language solutions have higher maintenance, configuration, documentation and test complexity
- More demanding on developers since they must know both languages
- Hybrid MATLAB/Python environments can be fragile. Example: after an OS security update I can no longer import geopandas in MATLAB
- Weigh the pro’s and con’s before using Python solutions in production!

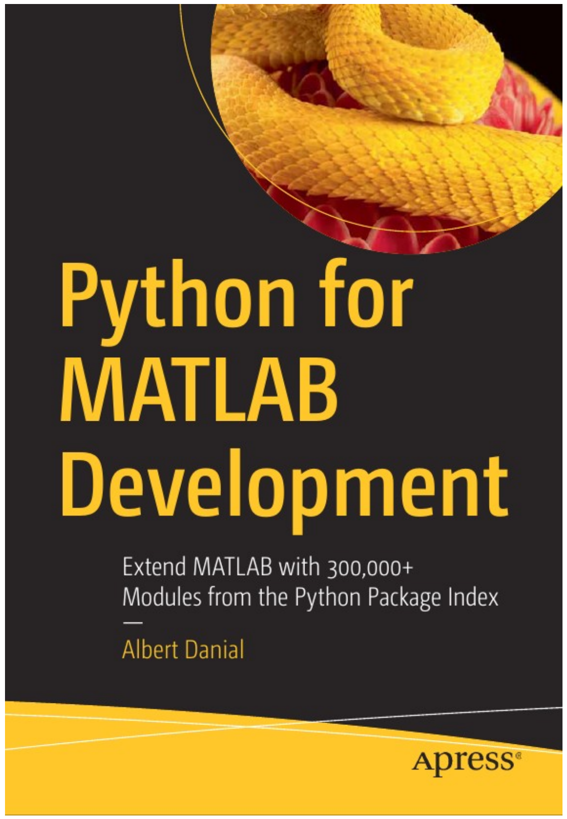
Summary

- Python can fill gaps in MATLAB’s capabilities
- MATLAB’s Python interface provides near-seamless access to Python modules, functions, and data types.

- MATLAB + Python = best of both worlds

Resources

- Examples shown in this presentation:
  https://github.com/Apress/python-for-matlab-development/tree/main/matlab_expo_2022
- `py2mat.m and mat2py.m:`
  https://github.com/Apress/python-for-matlab-development/tree/main/code/matlab_py
- The book
  - A comprehensive Python language tutorial using side-by-side examples with MATLAB
  - A guide to configuring a Python environment that pairs nicely with MATLAB
  - A collection of MATLAB-calling-Python recipes
  - Emphasizes scientific, numeric, and high performance computing


Python for MATLAB Development
Extend MATLAB with 300,000+ Modules from the Python Package Index
Albert Danial
Apress