# Python from MATLAB

Daniel Driver

E177-Advanced Matlab

April 21, 2015

# First Impressions

- Not available until 2014b
- More functionality in 2015a (according to the manual)
  - Index a List or Tuple with {}
- Pretty cumbersome at the moment
- No terrible though if you know python well
- Sometimes requires calling operator function instead of symbol
  - getitem instead of using "a=something[3]"
  - setitem instead of "something[3]=a"
- binary operations all seem to work though
  - e.g. +,-,*,>

# Example PyClass.py

## VehicleClass

```
class VehicleClass():
    def __init__(self,*argv):#constructor
        import numpy as np
        self.Velocity=np.array([0,0],dtype=float)
        try:
            self.NumPassengers=argv[1]
        except:
            self.NumPassengers=1
        self.StorageVolume=0.0
        self.Color=argv[0]
    def __add__(V1,V2): #overloads the '+' symbol
        a=VehicleClass(V1.Color,V1.NumPassengers+V2.NumPassengers)
        a.Velocity=V1.Velocity+V2.Velocity
        a.StorageVolume=V1.StorageVolume+V2.StorageVolume
        return a
    def __repr__(self):  #Like Matlab Display Function
        OutputString="""Vehicle with:
        Velocity {0},
        Storage Volume {1},
        Carries {2} Passangers
        and is the Color {3}""".format(self.Velocity,
                                       self.StorageVolume,
                                       self.NumPassengers,
                                       self.Color)
        return OutputString
    def accelerate(self,deltaV):
        self.Velocity+=deltaV
```

# Using a Python Class

## Python Usage

```
import PyClass import numpy as np
V1=PyClass.VehicleClass('Red')
V2=PyClass.VehicleClass('Yellow',5)

V1.accelerate(np.array([1,2]))
V2.accelerate(np.array([3,4]))

V3=V1+V2
V3.accelerate(np.array([5,1.3]))
print (V3)
multout=V3.Velocity*V1.Velocity
print('multout={0}'.format(multout))
```

- Do not do "import PyClass"
  - py.PyClass automatically imports PyClass
- Replace "from x import y"
  - "import x.y" in MATLAB

## Matlab Usage

```
V1=py.PyClass.VehicleClass('Red');
V2=py.PyClass.VehicleClass('Yellow',5);
V1.accelerate(py.numpy.array([1,2]));
V2.accelerate(py.numpy.array([3,4]));

V3=V1+V2;
V3.accelerate(py.numpy.array([5,1.3]))

py.print(V3)
display('multout=')
multout=V3.Velocity * V1.Velocity
```

- Pretty Clean MATLAB syntax
- "py" to first call a python Class
- after that mostly works
  - Only array inputs a little weird

# Trying to work with Numpy

- Have to use pure python operators to index
  - getitem
  - setitem
- Still able to do everything with native python commands

## MATLAB Usage

```
%used pure python operators to work with numpy array
test=py.numpy.random.random(py.tuple([5,5]))
py.operator.getitem(test,py.tuple([3,3]))
py.operator.setitem(test,py.tuple([0,1]),100.0)
%will return python values
value=py.operator.getitem(test,py.tuple([0,1]))
%can divide like noraal value/3.3
display('display test')
display(test)
display('vs py.print method')
py.print(test)
display('Transpose of T')
test.T
```

# Code That should work in 2015a

- Basic Indexing with {}
- Work with
  - Dictionary
  - Tuple
  - List

## MATLAB Usage

```
%put things in a dictionary
customers = py.dict
customers{'Smith'} = int32(2112);
customers{'Anderson'} = int32(3010);
customers{'Audrey'} = int32(4444);
customers{'Megan'} = int32(5000);
acct = customers{'Anderson'}

%make and loop over a list
li = py.list({1,2,3,4});
for n = li
disp(n{1})
end
```

# More 2015a

## MATLAB Usage

```
%indexing into string (1 indexed)
pstr = py.str('myfile');
pstr(1)

%tuple slicing (1indexed)
t = py.tuple({'a','bc',1,2,'def'});
t(1:2)
%list splicing matlab style
%   start:step:stop
%also 1 indexed
li = py.list({'a','bc',1,2,'def'});
li(1:2:end)

%can also get end
%get value
li{end}
%get python object of value
li(end)
```

## {} syntax

- seems to actually get value in matlab form
- only can get one value

## () syntax

- makes another python object
- can perform 1-d slicing

# Limitations-Indexing

You can access data in Python container objects, like lists and dictionaries, with index values, similar to referencing an element in a MATLAB matrix. There are, however, ways to index into matrices which are not supported for these Python types.

| Indexing Features Not Supported in MATLAB |
|---|
| Use of square brackets, `[]`. |
| Indexing into a container type that does not inherit from `collections.Sequence` or `collections.Mapping`. |
| Logical indexing. |
| Accessing data in a container with an arbitrary array of indices. An index must be of the form `start:step:stop`. |
| Comma-separated lists. |
| `numel` function does not return number of array elements. Returns 1. |

Figure : Current limitations of indexing Python Objects in MATLAB as of 2015a

# Limitations-General

| Features Not Supported in MATLAB |
| --- |
| Editing and reloading a Python module in the same MATLAB session. To use an updated module, restart MATLAB. |
| Closing the Python interpreter while running MATLAB. |
| Saving (serializing) Python objects into a MAT-file. |
| Interactive Python help (calling `py.help` without input arguments). |
| `py.input` and `py.raw_input` (version 2.7). |
| Accessing static properties of a Python class. |
| MATLAB `isa` function does not recognize virtual inheritance. |
| MATLAB class inheritance from a Python class. |
| Overloaded attribute access. |
| Nested Python classes. |
| Modules that start MATLAB in a separate process, for example, the `multiprocessing` module. |
| Modules that read `sys.argv`, the command-line arguments passed to a Python script, for example, `Tkinter`. |
| Dynamically generated Python classes, for example, `collections.namedtuple`. |
| Dynamically attaching new object attributes. Instead, use `py.setattr`. |
| Class names or other identifiers starting with an underscore (_) character. Instead, use the Python `py.getattr` and `py.setattr` functions. |

Figure : General Limitations as of MATLAB 2015a