Python for MATLAB® Users

Jian Tao
jtao@tamu.edu
Spring 2018 HPRC Short Course
02/16/2018
Relevant Short Courses and Workshops

Introduction to MATLAB Programming
https://hprc.tamu.edu/training/intro_matlab.html

Introduction to Python
https://hprc.tamu.edu/training/intro_python.html

Introduction to Scientific Python
https://hprc.tamu.edu/training/intro_scientific_python.html

Bring-Your-Own-Code Workshop
https://coehpc.engr.tamu.edu/byoc/
Offered regularly
MATLAB

MATLAB is a commercial software tool (language + IDE) for:

▪ Performing mathematical computations and signal processing
▪ Analyzing and visualizing data (excellent graphics tools)
▪ Modeling physical systems and phenomena
▪ Testing engineering designs
▪ Functionalities enhanced via apps & packages
Python

Python is an open source **programming language** that is empowered by a large selection of open source libraries:

- With NumPy, one can operate large, multi-dimensional arrays and matrices.
- With Matplotlib, one can visualize various types of data.
- With TensorFlow, one can build applications with machine intelligence.
- With Django, one can build sophisticated web sites.
- and many more...

[https://www.python.org/about/gettingstarted/](https://www.python.org/about/gettingstarted/)
Round One: Python vs MATLAB

Matlab ecosystem

- Core Matlab
  - IDE
  - interpreter
  - programming lang.
  - HUGE standard library
  - GUI builder (sort of)
- File exchange
- Simulink
- Image processing TK
- etc.

Python ecosystem

- Core Python
  - interpreter
  - programming lang.
  - standard library
- IDE's:
  - Pyzo
  - Spyder
  - Ninja IDE
  - Wing
  - PyCharm
- Numpy
- Matplotlib
- Scipy
- PyQt4
- etc.

http://www.pyzo.org
Python
Free & open source for both core & extra libraries.

Relatively small community in science & engineering, but rapidly growing.

Installation of different packages to set up a functional development environment which is not on par with that of MATLAB so far.

MATLAB
Expensive core & extra packages & proprietary algorithms.

Large scientific community & legacy code base.

Single installation with a user friendly Integrated Development Environment & wonderful plotting tools.

http://www.pyzo.org
Arithmetic Operators

+ Addition
- Subtraction
/ division
% mod
* multiplication
// integer division
** to the power of
Arithmetic Operators and Order of Operations

- Addition (+), Subtraction (-), Multiplication (*), Division (/), Power (**)
- Order of Operations (same rules you should already know from math class and using a calculator)
  1. Complete all calculations inside parentheses or brackets using the precedent rules below
  2. Powers (left to right)
  3. Multiplication and Division (left to right)
  4. Addition and Subtraction (left to right)
Arithmetic Expressions

Some examples:

```python
>>> 10/5*2
>>> 5*2**3+4*2  #^ -> **
>>> -1**4
>>> 8**1/3
>>> pi        #pi is not defined
```
Relational Operators

== True, if it is equal
!= True, if not equal to
< less than
> greater than
<= less than or equal to
>= greater than or equal to
Boolean and Bitwise Operators

- **and**: Logical and
- **or**: Logical or
- **not**: Not
- **^**: Exclusive OR
- **|**: Bitwise OR
- **~**: Negate
- **&**: Bitwise And
- **>>**: Right shift
- **<<**: Left shift
The basic types of Python include float, int, str, complex, and bool. A variable can be deleted with del

```python
>>> b = True
>>> whos          # works on iPython!
>>> type(b)       # type of the variable
>>> x = "Hi"
>>> y = 10
>>> z = complex(1, 2)
>>> print(b, x, y, z)
>>> del b; print(b)
```
Naming Rules for Variables

- Variable names must begin with a letter
  >>>4c = 12
- Names can include any combinations of letters, numbers, and underscores
  >>>c_4 = 12
- Maximum length for a variable name is not limited
- Python is case sensitive. The variable name A is different than the variable name a.
Exercise

Create two variables: \( a = 4 \) and \( b = 17.2 \)

Now use Python to perform the following set of calculations:

\[
(b+5.4)^{1/3} \quad b^2-4b+5a \quad a>b \text{ and } a>1.0 \quad a!=b
\]
Basic Syntax for Statements

1. No spaces or tab are allowed at the start of a statement.
2. Comments start with '#' 
3. Statements finish at the end of the line ('\' can be used to indicate an unfinished line)

```python
>>> 1+2
>>> name = 'Adam'       # Indentation Error!
>>> months = "Aug\n..."
```
Displaying Variables

We can display a variable (i.e., show its value) by simply typing the name of the variable at the command prompt (leaving off the semicolon).

We can also use `print` to display variables. Type the following commands at the command prompt:

```python
>>>print('The value of x is:'); print(x)
```
## Numerical Data Types - NumPy

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool_</td>
<td>Boolean (True or False) stored as a byte</td>
</tr>
<tr>
<td>int_</td>
<td>Default integer type (same as C long; normally either int64 or int32)</td>
</tr>
<tr>
<td>intc</td>
<td>Identical to C int (normally int32 or int64)</td>
</tr>
<tr>
<td>intp</td>
<td>Integer used for indexing (same as C ssize_t; normally either int32 or int64)</td>
</tr>
<tr>
<td>int8</td>
<td>Byte (-128 to 127)</td>
</tr>
<tr>
<td>int16</td>
<td>Integer (-32768 to 32767)</td>
</tr>
<tr>
<td>int32</td>
<td>Integer (-2147483648 to 2147483647)</td>
</tr>
<tr>
<td>int64</td>
<td>Integer (-9223372036854775808 to 9223372036854775807)</td>
</tr>
<tr>
<td>uint8</td>
<td>Unsigned integer (0 to 255)</td>
</tr>
<tr>
<td>uint16</td>
<td>Unsigned integer (0 to 65535)</td>
</tr>
<tr>
<td>uint32</td>
<td>Unsigned integer (0 to 4294967295)</td>
</tr>
<tr>
<td>uint64</td>
<td>Unsigned integer (0 to 18446744073709551615)</td>
</tr>
<tr>
<td>float_</td>
<td>Shorthand for float64.</td>
</tr>
<tr>
<td>float16</td>
<td>Half precision float: sign bit, 5 bits exponent, 10 bits mantissa</td>
</tr>
<tr>
<td>float32</td>
<td>Single precision float: sign bit, 8 bits exponent, 23 bits mantissa</td>
</tr>
<tr>
<td>float64</td>
<td>Double precision float: sign bit, 11 bits exponent, 52 bits mantissa</td>
</tr>
<tr>
<td>complex_</td>
<td>Shorthand for complex128.</td>
</tr>
<tr>
<td>complex64</td>
<td>Complex number, represented by two 32-bit floats (real and imaginary components)</td>
</tr>
<tr>
<td>complex128</td>
<td>Complex number, represented by two 64-bit floats (real and imaginary components)</td>
</tr>
</tbody>
</table>
Why should I care how data is stored in a computer?

Perform each of the following calculations in your head.

\[ a = \frac{4}{3} \]
\[ b = a - 1 \]
\[ c = 3 \times b \]
\[ e = 1 - c \]

What does Python get?
Why should I care how data is stored in a computer?

What does Python get?

```python
>>> a = 4/3  # 1.3333333333333333
>>> b = a - 1  # 0.33333333333333326
>>> c = 3*b  # 0.99999999999999998
>>> e = 1 - c  # 2.220446049250313e-16
```

It is not possible to perfectly represent all real numbers using a finite string of 1s and 0s.
Creating Strings (Text Variables)

Single quotes & double quotes are the same in Python. '\' can be used if there is a quote in a string.

```python
>>> month = 'Aug'
>>> my_str1 = 'Tom\'s toy'
>>> my_str2 = "Tom's toy"
```
## Some Useful Math Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Python</th>
<th>Function</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>cosine</td>
<td>cos</td>
<td>square root</td>
<td>sqrt</td>
</tr>
<tr>
<td>sine</td>
<td>sin</td>
<td>exponential</td>
<td>exp</td>
</tr>
<tr>
<td>tangent</td>
<td>tan</td>
<td>logarithm (base 10)</td>
<td>log10</td>
</tr>
<tr>
<td>arc cosine</td>
<td>acos</td>
<td>natural log (base e)</td>
<td>log</td>
</tr>
<tr>
<td>arc sine</td>
<td>asin</td>
<td>round to nearest integer</td>
<td>round</td>
</tr>
<tr>
<td>arc tangent</td>
<td>atan</td>
<td>round down to integer</td>
<td>floor</td>
</tr>
<tr>
<td>Euclidean norm</td>
<td>hypot</td>
<td>round up to integer</td>
<td>ceil</td>
</tr>
</tbody>
</table>

**Note:** All the functions are in the math package, which must be imported before these functions can be used. Trigonometric functions assume input in radians. Euclidean norm returns $\sqrt{x^2 + y^2}$. 
Help

- The `help` command provides information about a function. Type `help("math.cos")` at the command prompt. This only works if you know the name of the function you want help with.
- Type `help()` to get the interactive help utility.
  
  `help>math.cos`
Data Structures: Tuple, List, Set, & Dictionary

Image Credit: https://medium.com/@meghamohan
Lists & Tuples

- **Lists** are the most commonly used data structure.
- A sequence of data that is enclosed in square brackets.
- Each element can be accessed by calling its index starting with 0.
- Lists are declared with `list()` or `[]`.

- **Tuples** are similar to lists.
- The elements in a list can be changed (mutable), but in tuple they can not be changed (immutable).
- Each element can be accessed by calling its index starting with 0.
- Tuples are declared with `tuple()` or `()`.
Lists & Tuples

List examples:

```python
>>> months=['Oct','Nov','Dec']
>>> names=list()
>>> type(months)
>>> months[0]
>>> months[-1] #try others
>>> months[0] = 'Sep'
>>> months
>>> m=tuple(months)
```

Tuple examples:

```python
>>> months=('Oct','Nov','Dec')
>>> names=tuple()
>>> type(months)
>>> months[0]
>>> months[-1] #try others
>>> months[0] = 'Sep'
>>> months
>>> m=list(months)
```
Sets

- Sets are mainly used to eliminate repeated numbers in a sequence/list.
- It is also used to perform some standard set operations.
- Sets are declared with `set()` or `{}`.
- Also `set([sequence])` can be executed to declare a set with elements.

Examples:

```python
>>> months={'Oct','Nov','Dec','Dec'}
>>> names=set()
>>> type(months)
>>> months[0]    # error!
>>> months.add('Sep')
>>> m=list(months)
```
Dictionaries

- **Dictionaries** are mappings between keys and items stored in the dictionaries.
- Alternatively one can think of dictionaries as sets in which something stored against every element of the set.
- Dictionaries are mutable.
- To define a dictionary, equate a variable to `{}` or `dict()`

Examples:

```python
>>> months={'Oct':'October','Nov':'November','Dec':'December'}
>>> names=dict()
>>> type(months)
>>> months['Oct']
>>> months.update({'Sep':'September'})
>>> months['Oct']='Current Month'
>>> m=list(months)
```
Conditional Statements & Loops
Loop Control Statements - *for*

*for* statements help repeatedly execute a block of code for a certain number of iterations.

**Matlab**

```matlab
x = zeros(1,10);
for n = 1 : 10
    x(n) = n;
end
```

**Python**

```python
import numpy as np
x = np.zeros(10)
for n in range(10):
    x[n] = n
```
Loop Control Statements - *while*

*while* statements repeatedly execute a block of code as long as a condition is satisfied.

%%%matlab
n = 1;
sum = 0;
while n <= 100
    sum = sum + n;
n = n + 1;
end

%%%python
n = 1
sum = 0
while n <= 100:
    sum = sum + n
    n = n + 1
Conditional Statements

Execute statements if condition is true

%matlab
if a>10
    disp('a > 10');
elseif a<10
    disp('a < 10')
else
    disp('a = 10')
end

#python
if a>10:
    print('a > 10')
elif a<10:
    print('a < 10')
else:
    print('a = 10')
Nested-Loop: Simple Example

%matlab
for r = 1:4
    for c = 1:4
        fprintf('(%i,%i)\n',r,c);
    end
end

#python
for r in range(1,5):
    for c in range(1,5):
        print('(%i,%i)'%(r, c))
Adding Break Statements

What if we add a break statement in the outer loop?

%matlab
for r = 1:4
    if r == 2
        break;
    end
    for c = 1:4
        fprintf('(%i,%i)\n',r,c);
    end
end

#python
for r in range(1,5):
    if r == 2:
        break
    for c in range(1,5):
        print('(%i,%i)'%(r, c))
Adding Break Statements

What if we add a break statement in the inner loop?

%matlab
for r = 1:4
    end
    for c = 1:4
        if r == 2
            break;
        fprintf('(%i,%i)\n',r,c);
        end
end

#python
for r in range(1,5):
    for c in range(1,5):
        if r == 2:
            break
        print('(%i,%i)'%(r,c))
Exercise: Output

Write a script that will display each of the following shapes using asterisks *

```
*****  *****  *
*****  *  *
*****  *  *
*****  *  *
*****  *****
```

Solid Square  Open Square  Triangle
Function

%Matlab
function [out]=myabs(number)
    if number > 0
        out = number
    else
        out = -number
    end
end

#Python
def myabs(number):
    if number > 0:
        return number
    else:
        return -number
NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities
Create an Array

There are multiple ways to create an array with numpy.

```python
>>> import numpy as np
>>> a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> b = np.array([[1, 2, 3], [4, 9, 6], [7, 8, 9]])
>>> c = np.zeros((3, 3))
>>> d = np.ones((3, 3))
>>> f = np.arange(0, 100, 10)
>>> g = np.random.uniform(0, 1, (3, 5))
>>> h = np.eye(5)
```
Array & Matrix Operations - II

>> %matlab
>> b + 10
>> sin(b)
>> b'
>> inv(b)
>> b*inv(b)
>> b.*b
>> b.^2

>>> #python
>>> b + 10 #each element + 10
>>> np.sin(b) #sin function
>>> b.transpose() #transpose
>>> c=np.linalg.inv(b) #inverse
>>> np.dot(b,c) #matrix multiplication
>>> b*c #element-wise multiplication
>>> b**2 #element-wise square
Array & Matrix Operations - III

Numpy arrays can be concatenated

```matlab
>> B_H = [b, b] %horizontal
>> B_V = [b; b] %vertical
```

```python
>>> B_H = np.concatenate((b, b), axis=1)
>>> B_V = np.concatenate((b, b), axis=0)
```

Python index goes starting from 0 while MATLAB index starting from 1.

```matlab
>> g(3, 5)
>> g(1:3, 5)
>> g(3, :)
```

```python
>> g[2, 4]
>> g[0:2, 4]
>> g[2, :]
```
Plots with Matplotlib
Matplotlib is a widely used Python plotting library.

```python
>>> import numpy as np
>>> x = np.arange(0, np.pi*2, np.pi/100)
>>> y = np.sin(x)
>>> plt.plot(x, y)
>>> plt.show()
```
Simple Surface Plot
Surface plot typically display a surface defined by a function in two variables, $z = f(x, y)$.

```
>>%matlab
>>[x,y] = meshgrid(-2:.2:2);
>>z = x .* exp(-x.^2 - y.^2);
>>figure %new figure window
>>surf(x,y,z)
```

```
>>>#python
>>>from mpl_toolkits.mplot3d import Axes3D
>>>import matplotlib.pyplot as plt
>>>from matplotlib import cm
>>>fig = plt.figure()
>>>ax = fig.gca(projection='3d')
>>>x, y = np.meshgrid(np.arange(-2, 2, 0.2), np.arange(-2, 2, 0.2))
>>>z = x*np.exp(-(x**2 + y**2))
>>>surf = ax.plot_surface(x, y, z, cmap=cm.viridis)
>>>plt.show()
```
Online Resources

Official Python Tutorial
https://docs.python.org/3.6/tutorial/

NumPy for MATLAB users
http://mathesaurus.sourceforge.net/matlab-numpy.html

Matplotlib Gallery
https://matplotlib.org/gallery.html

Gallery of Jupyter Notebooks

Introduction to Python 3 Notebooks
https://gitlab.erc.monash.edu.au/andrease/Python4Maths/tree/master
Acknowledgements

- The slides are created based on the educational materials for MATLAB from Kathleen Ossman and Gregory Bucks under BSD license.
- Supports from Texas A&M Engineering Experiment Station (TEES) and High Performance Research Computing (HPRC).
Appendix
When you press a key on your computer keyboard, the key that you press is translated to a binary code.

<table>
<thead>
<tr>
<th>Character</th>
<th>Binary Code</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10000001</td>
<td>65</td>
</tr>
<tr>
<td>a</td>
<td>11000001</td>
<td>97</td>
</tr>
<tr>
<td>0</td>
<td>01100000</td>
<td>48</td>
</tr>
</tbody>
</table>
# ASCII Code

ASCII stands for American Standard Code for Information Interchange.
Terminology

A **bit** is short for **binary digit**. It has only two possible values: On (1) or Off (0).

A **byte** is simply a string of 8 bits.

A **kilobyte** (KB) is 1,024 ($2^{10}$) bytes.

A **megabyte** (MB) is 1,024 KB or $1,024^2$ bytes.

A **gigabyte** (GB) is 1,024 MB or $1,024^3$ bytes.
How Computers Store Variables

Computers store all data (numbers, letters, instructions, …) as strings of 1s and 0s (bits).

A bit is short for binary digit. It has only two possible values: On (1) or Off (0).
Data Types: double and single

- A double uses 64 bits to store a real number.
- A single uses 32 bits to store a real number. Python does not support single by default though Numpy supports it.
- Doubles and singles can be used to represent both integers and non-integers.