Introduction to Python

Yali Bian, Srijith Rajamohan

Advanced Research Computing, Virginia Tech

Wednesday 6th December, 2017

Course Contents

This Presentation:

- Introduction to Python
- Python Programming

Section 1

Python Features

Why Python ?

- Interpreted
- Intuitive and minimalistic code
- Expressive language
- Dynamically typed
- Automatic memory management

Python Features

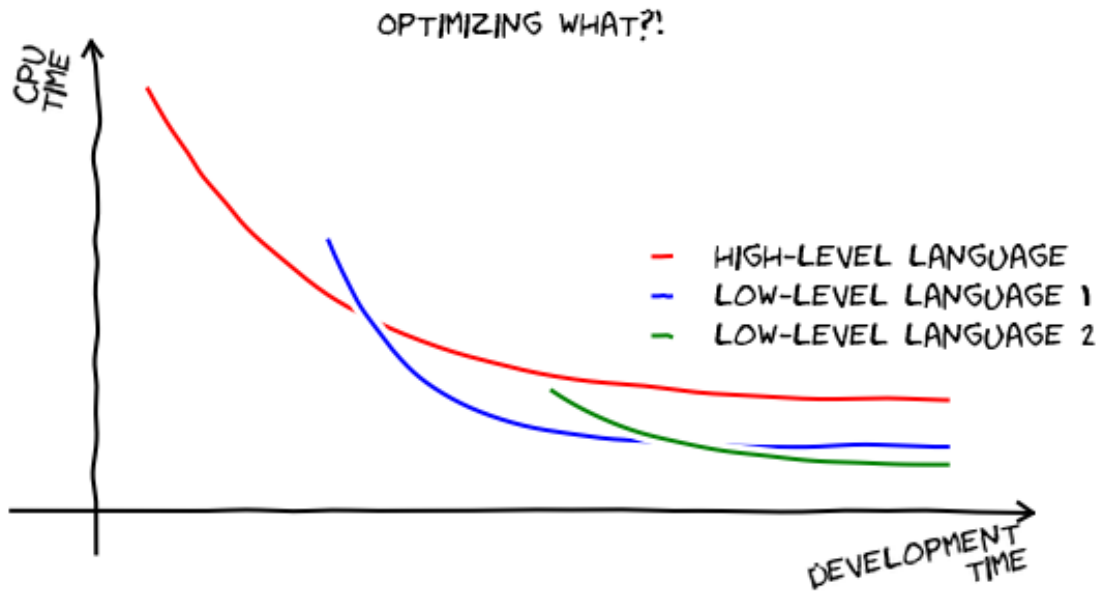
Advantages

- Ease of programming
- Minimizes the time to develop and maintain code
- Modular and object-oriented
- Large community of users
- A large standard and user-contributed library

Disadvantages

- Interpreted and therefore slower than compiled languages
- Decentralized with packages

Code Performance vs Development Time



Versions of Python

- Two versions of Python in use - Python 2 and Python 3
- Python 3 not backward-compatible with Python 2
- A lot of packages are available for Python 2
- Machine Learning crowd has adopted Python 3
- **Python 2 supposedly no longer developed after 2020**
- Check version using the following command

Example

```
$ python --version
```

Anaconda Python

- Anaconda Python is a free Python distribution
- Used for data analytics, scientific computing
- Conda - an open-source package and environment manager
- Has both Python 2 and 3 options
- Launch the anaconda app and select the Ipython interface

GUI and Notebook

Spyder:

- an IDE for Python - coding, debugging and execution in an integrated environment.
- Code editor with syntax highlighting
- Variable explorer

Jupyter Notebook

- Command history
- Execute system commands
- Command auto-completion
- Great for plotting!
- `http://ipython.org`

Anaconda Python - conda

Example

```
# Search for a package type
$ binstar search -t conda ggplot2
...
# Install package using:
$ conda install --channel https://conda.
    anaconda.org/r r-ggplot2
...
# You can also use the following
$ pip install <pkg name>
```

Aside: Notation

We will use the following notation in these slides:

Example (Command Line)

```
$ python hello.py
```

Example (Python Interpreter)

```
>>> print ("Hello world!")
```

Hello World - hello.py !

NOTE: **Indentation** is very important in Python. It defines the extent of a code block.
Let us look at the file 'hello.py'

Example

```
#!/usr/bin/env python
# Path to python interpreter on Unix
    systems

print("Hello World!")
```

Python Interpreter

To run a program named 'hello.py' on the command line

Example

```
$ python hello.py
```

You can do the same in the interpreter. Invoke the interpreter by typing 'python' on the command line and then use `execfile`

Example

```
>>> execfile("hello.py")
```

Python Modules

- Python functionality such as I/O, string manipulation, math routines etc. provided by modules
- Reference to Python 2 standard library of modules at `http://docs.python.org/2/library/`
- ditto for Python 3 `https://docs.python.org/3/library/`

Python Modules

Example

```
import math  #This imports the whole
   module
x = math.sin( math.pi )
print x
```

Example

```
from math import *  #This imports
# all symbols to the current namespace
x = sin( pi )
print x
```

Python Modules - Documentation

- Your Python installation already comes with plenty of modules built-in
- Use the `dir` command to list the symbols (functions, classes and variables) in a module
- The `help` command can be used on each function to obtain documentation as long as they have 'docstrings', which is a string within triple quotes

Example

```
def test_help():
        """Prints 'hello'."""
        print "hello"
```

Python Modules - Documentation

Example

```
>>> print(dir(math))
['__doc__', '__loader__', '__name__', '
    __package__', 'acos', 'acosh', 'asin',
    'asinh', 'atan', 'atan2', 'atanh', '
    ceil', 'copysign', 'cos', 'cosh', '
    degrees', 'e', 'erf', 'erfc', 'exp', '
    expm1', 'fabs', 'factorial', 'floor', '
    fmod', 'frexp', 'fsum', 'gamma', 'hypot
    ', 'isfinite', 'isinf', 'isnan', 'ldexp
    ', 'lgamma', 'log', 'log10', 'log1p', '
    log2', 'modf', 'pi', 'pow', 'radians',
    'sin', 'sinh', 'sqrt', 'tan', 'tanh', '
    trunc']
```

Python Modules - Documentation

Example

```
>>> help(math.log)
Help on built-in function log in module
    math:

log(...)
    log(x[, base])

    Return the logarithm of x to the given
        base.
    If the base not specified, returns the
        natural logarithm (base e) of x.
```

Directory and file operation

Get current directory

Example

```
>>> import os
>>> os.getcwd()
'/home/srijithr'
>>> os.curdir
'.'
>>> os.listdir(os.curdir)
['.index.rst.swo',
  'control_flow.rst',
 'debugging.rst',
 ...
```

Directory and file operation

Example

```
>>> os.path.split(a)
 ('/home/srijithr','junk.txt')
>>> os.path.dirname(a)
('/home/srijithr'
>>> os.path.basename(a)
 'junk.txt'
>>> os.path.splitext(os.path.basename(a))
('junk', '.txt')
>>> os.path.exists('junk.txt')
True
>>> os.path.isfile('junk.txt')
True
>>> os.path.isdir('junk.txt')
False
```

Section 2

Variables

- Variable names can contain alphanumerical characters and some special characters
- It is common to have variable names start with a lower-case letter and class names start with a capital letter
- Some keywords are reserved such as `'and'`, `'assert'`, `'break'`, `'lambda'`. A list of keywords are located at `https://docs.python.org/2.5/ref/keywords.html`
- Python is dynamically typed, the type of the variable is derived from the value it is assigned.
- A variable is assigned using the '=' operator

Variable naming

Variable names can make it easier (or harder) to understand a program!
Try to give variables clear, descriptive names!

Example

```
log_file = open("/var/log/syslog", "r")
userName = "pradics"
```

Avoid single-character names and abbreviations!

Example

```
f = open("/var/log/syslog", "r")
un = "pradics"
```

Variable types

- Variable types
  - Integer (`int`)
  - Float (`float`)
  - Boolean (`bool`)
  - Complex (`complex`)
  - String (`str`)
  - ...
  - User Defined! (classes)
- Documentation
  - https://docs.python.org/3/library/types.html
  - https://docs.python.org/3/library/datatypes.html

Variable types

- Use the type function to determine variable type

Example

```
>>> log_file = open("/home/srijithr/
    logfile","r")
>>> type(log_file)
file
```

Variable types

- Variables can be *cast* to a different type

Example

```
>>> share_of_rent = 295.50 / 2.0
>>> type(share_of_rent)
float
>>> rounded_share = int(share_of_rent)
>>> type(rounded_share)
int
```

Operators

- Arithmetic operators +, -, *, /, // (integer division for floating point numbers), '**' power
- Boolean operators `and`, `or` and `not`
- Comparison operators >, <, >= (greater or equal), <= (less or equal), == equality

Operators

Example

```
>>> bar_tab = 35.28
>>> my_share = bar_tab / 3
>>> tip_amount = my_share * 0.2
>>> my_total = my_share + tip_amount
>>> enough_money = my_total < 15.00
>>> feeling_good = True
>>> good_night = enough_money and
    feeling_good
>>> print(my_total)
14.112
>>> print(enough_money)
True
>>> print(good_night)
True
```

Strings (str)

Example

```
>>> dir(str)
[..., 'capitalize', 'center', 'count', '
    decode', 'encode', 'endswith', '
    expandtabs', 'find', 'format', 'index',
     'isalnum', 'isalpha', 'isdigit', '
    islower', 'isspace', 'istitle', '
    isupper', 'join', 'ljust', 'lower', '
    lstrip', 'partition', 'replace', 'rfind
    ', 'rindex', 'rjust', 'rpartition', '
    rsplit', 'rstrip', 'split', 'splitlines
    ', 'startswith', 'strip', 'swapcase', '
    title', 'translate', 'upper', 'zfill']
```

Strings

Example

```
>>> greeting = "Hello world!"
>>> len(greeting)
12
>>> greeting
'Hello world'
>>> greeting[0] # indexing starts at 0
'H'
>>> greeting.replace("world", "test")
Hello test!
```

Printing strings

Example

```
# concatenates strings with a space
>>> print("Go", "Hokies")
Go Hokies
# concatenated without space
>>> print("Go" + "Tech" + "Go")
GoTechGo
# C-style string formatting
>>> print("Bar Tab = %f" %35.28)
Bar Tab = 35.280000
# Creating a formatted string
>>> total = "My Share = %.2f. Tip = %d" %
    (11.76, 2.352)
>>> print(total)
My Share = 11.76. Tip = 2
```

Lists

Array of elements of arbitrary type

Example

```
>>> numbers = [1,2,3]
>>> type(numbers)
list
>>> arbitrary_array = [1,numbers,"hello"]
>>> type(arbitrary_array)
list
```

Lists

Example

```
# create a new empty list
>>> characters = []
# add elements using 'append'
>>> characters.append("A")
>>> characters.append("d")
>>> characters.append("d")
>>> print(characters)
['A', 'd', 'd']
```

Lists

Lists are *mutable* - their values can be changed.

Example

```
>>> characters = ["A","d","d"]
# Changing second and third element
>>> characters[1] = "p"
>>> characters[2] = "p"
>>> print(characters)
['A', 'p', 'p']
```

Lists

Example

```
>>> characters = ["A","d","d"]
# Inserting before "A","d","d"
>>> characters.insert(0, "i")
>>> characters.insert(1, "n")
>>> characters.insert(2, "s")
>>> characters.insert(3, "e")
>>> characters.insert(4, "r")
>>> characters.insert(5, "t")
>>>print(characters)
['i', 'n', 's', 'e', 'r', 't', 'A', 'd', '
   d']
```

Lists

Example

```
>>> characters = ['i', 'n', 's', 'e', 'r',
    't', 'A', 'd', 'd']
# Remove first occurrence of "A" from list
>>> characters.remove("A")
>>> print(characters)
['i', 'n', 's', 'e', 'r', 't', 'd', 'd']
# Remove an element at a specific location
>>> del characters[7]
>>> del characters[6]
>>> print(characters)
['i', 'n', 's', 'e', 'r', 't']
```

Tuples

Tuples are like lists except they are *immutable*. Difference is in performance

Example

```
>>> point = (10, 20)  # Note () for tuples
    instead of []
>>> type(point)
tuple
>>> point = 10,20
>>> type(point)
tuple
>>> point[2] = 40 # This will fail!
TypeError: 'tuple' object does not support
    item assignment
```

Dictionary

Dictionaries are lists of key-value pairs

Example

```
>>> prices = {"Eggs" : 2.30,
...           "Sausage" : 4.15,
...           "Spam" : 1.59,}
>>> type(prices)
dict
>>> print (prices)
{'Eggs': 2.3, 'Sausage': 4.15, 'Spam':
   1.59}
>>> prices["Spam"]
1.59
```

File I/O

- File modes denote how files are opened
- r for read-only mode
- w for write-only mode, this can overwrite existing files
- a for appending to a file
- r+ for read and write
- b for binary mode (in addition to one of the other modes)

File I/O

To write to a file use the following

Example

```
>>> work = open('workfile', 'w') # opens
    the workfile file
>>> type(work)
file
>>> work.write('Teach a python tutorial.')
>>> work.write('Be awesome.')
>>> work.close()
```

File I/O

To read from a file use the following

Example

```
>>> work = open('workfile', 'r')
>>> task = work.read()
>>> print(task)
Teach a python tutorial.
>>> task2 = work.read()
>>> print(task2)
Be awesome.
>>> work.close()
```

Conditional statements: if, elif, else

Example

```
>>> I_am_tired = False
>>> I_am_hungry = True
>>> if I_am_tired is True:    # Note the
    colon for a code block
...     print ("You have to teach!")
... elif I_am_hungry is True:
...     print ("No food for you!")
... else:
...     print "Go on...!"
...
No food for you!
```

Loops - For

Example

```
>>> for i in [1,2,3]: # i is an arbitrary
    variable for use within the loop
    section
...     print(i)
1
2
3
>>> for word in ["scientific", "computing"
    , "with", "python"]:
...     print(word)
scientific
computing
with
python
```

Loops - While

Example

```
>>>i = 0
>>>while i < 5:
...     print(i)
...     i = i + 1
0
1
2
3
4
```

Functions

Example

```
>>> def print_word_length(word):
...     """
...     Print a word and how many
    characters it has
...     """
...     print(word + " has " + str(len(
    word)) + " characters.")
>>>print_word_length("Diversity")
Diversity has 9 characters.
```

Functions - arguments

- Passing immutable arguments like integers, strings or tuples acts like *call-by-value*
  - They cannot be modified!
- Passing mutable arguments like lists behaves like *call-by-reference*

Functions - arguments

Call-by-value

Example

```
>>> def make_me_rich(balance):
            balance = 1000000
account_balance = 500
>>> make_me_rich(account_balance)
>>> print(account_balance)
500
```

Functions - arguments

Call-by-reference

Example

```
>>> def talk_to_advisor(tasks):
            tasks.insert(0, "Publish")
            tasks.insert(1, "Publish")
            tasks.insert(2, "Publish")
>>> todos = ["Graduate","Get a job","...",
   "Profit!"]
>>> talk_to_advisor(todos)
>>> print(todos)
 ["Publish","Publish","Publish","Graduate"
    ,"Get a job","...","Profit!"]
```

Functions - arguments

- However, you cannot assign a new object to the argument
  - A new memory location is created for this list
  - This becomes a local variable

Example

```
>>> def switcheroo(favorite_teams):
...     print (favorite_teams)
...     favorite_teams = ["Redskins"]
...     print (favorite_teams)
>>> my_favorite_teams = ["Hokies", "
   Nittany Lions"]
>>> switcheroo(my_favorite_teams)
["Hokies", "Nittany Lions"]
["Redskins"]
>>> print (my_favorite_teams)
["Hokies", "Nittany Lions"]
```

Functions - Multiple Return Values

Example

```
>>> def powers(number):
...     return number ** 2, number ** 3
>>> squared, cubed = powers(3)
>>> print(squared)
9
>>> print(cubed)
27
```

Functions - Default Values

Example

```
>>> def likes_food(person, food="Broccoli"
    , likes=True):
...     if likes:
...         print(str(person) + " likes "
    +  food)
...     else:
...         print(str(person) + " does not
    like " +  food)
>>> likes_food("Srijith", likes=False)
Srijith does not like Broccoli
```

Classes

- Classes are one of the key features of object-oriented programming
- An instance of a class is an object
- A class contains attributes and methods that are associated with this object

Classes

Example

```
>>> class Point:
...     def __init__(self, x, y):
...         self.x = x
...         self.y = y
...
...     def translate(self, dx, dy):
...         self.x += dx
...         self.y += dy
...
...     def __str__(self):
...         return("Point at [%f, %f]" % (
   self.x, self.y))
```

Classes

Example

```
# To create a new object
>>> origin = Point(0, 0) # this will
    invoke the __init__ method in the Point
     class
>>> print(origin)              # this will
    invoke the __str__ method
Point at [0.000000, 0.000000]
```

Section 3

NumPy

Used in almost all numerical computations in Python

- Used for high-performance vector and matrix computations
- Provides fast precompiled functions for numerical routines
- Written in C and Fortran
- Vectorized computations

Why NumPy?

Example

```
>>> from numpy import *
>>> import time
>>> def trad_version():
      t1 = time.time()
      X = range(10000000)
      Y = range(10000000)
      Z = []
      for i in range(len(X)):
        Z.append(X[i] + Y[i])
      return time.time() - t1

>>> trad_version()
1.9738149642944336
```

Why NumPy?

Example

```
>>> def numpy_version():
        t1 = time.time()
        X = arange(10000000)
        Y = arange(10000000)
        Z = X + Y
        return time.time() - t1

>>> numpy_version()
 0.059307098388671875
```

Arrays

Example

```
>>> from numpy import *
#  the argument to the array function is a
    Python list
>>> v = array([1,2,3,4])
#  the argument to the array function is a
    nested Python list
>>> M = array([[1, 2], [3, 4]])
>>> type(v), type(M)
(numpy.ndarray, numpy.ndarray)
```

Arrays

Example

```
>>> v.shape, M.shape
((4,), (2, 2))
>>> M.size
4
>>> M.dtype
dtype('int64')
# Explicitly define the type of the array
>>> M = array([[1, 2], [3, 4]], dtype=
    complex)
```

Arrays - Using array-generating functions

Example

```
>>> x = arange(0, 10, 1) # arguments:
    start, stop, step
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> linspace(0,10,11) # arguments: start,
    end and number of points ( start and
    end points are included )
array([  0.,   1.,   2.,   3.,   4.,   5.,
        6.,   7.,   8.,   9.,  10.])
```

Mgrid

Example

```
>>> x,y = mgrid[0:3,0:2]
>>> x
array([[0, 0],
       [1, 1],
       [2, 2]])
>>> y
array([[0, 1],
       [0, 1],
       [0, 1]])
```

Diagonal and Zero matrix

Example

```
>>> diag([1,2,3])
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
>>> zeros((3,3))
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

Array Access

Example

```
>>> M = random.rand(3,3) # not a Numpy
    function
>>> M
array([
[ 0.37389376,  0.64335721,  0.12435669],
[ 0.01444674,  0.13963834,  0.36263224],
[ 0.00661902,  0.14865659,  0.75066302]])
>>> M[1,1]
0.13963834214755588
```

Array Access

Example

```
# Access the first row
>>> M[1]
array(
[ 0.01444674,  0.13963834,  0.36263224])
# The first row can be also be accessed
   using this notation
>>> M[1,:]
array(
[ 0.01444674,  0.13963834,  0.36263224])
# Access the first column
>>> M[:,1]
array(
[ 0.64335721,  0.13963834,  0.14865659])
```

Array Access

Example

```
# You can also assign values to an entire
    row or column
>>> M[1,:] = 0
>>> M
array([
[ 0.37389376,  0.64335721,  0.12435669],
[ 0.        ,  0.        ,  0.        ],
[ 0.00661902,  0.14865659,  0.75066302]])
```

Array Slicing

Example

```
# Extract slices of an array
>>> M[1:3]
array([
[ 0.        ,  0.        ,  0.        ],
[ 0.00661902,  0.14865659,  0.75066302]])
>>> M[1:3,1:2]
array([
[ 0.        ],
[ 0.14865659]])
```

Array Slicing - Negative Indexing

Example

```
# Negative indices start counting from the
    end of the array
>>> M[-2]
array(
[ 0.,  0.,  0.])
>>> M[-1]
array(
[ 0.00661902,  0.14865659,  0.75066302])
```

Array Access - Strided Access

Example

```
# Strided access
>>> M[::2,::2]
array([[ 0.37389376,  0.12435669],
       [ 0.00661902,  0.75066302]])
```

Array Operations - Scalar

These operation are applied to all the elements in the array

Example

```
>>> M*2
array([
[ 0.74778752,  1.28671443,  0.24871338],
[ 0.        ,  0.        ,  0.        ],
[ 0.01323804,  0.29731317,  1.50132603]])
>>> M + 2
array([
[ 2.37389376,  2.64335721,  2.12435669],
[ 2.        ,  2.        ,  2.        ],
[ 2.00661902,  2.14865659,  2.75066302]])
```

Matrix multiplication

Example

```
>>> M * M # Element-wise multiplication
array([
[1.397965e-01,4.139085e-01,1.546458e-02],
[0.000000e+00,0.000000e+00,0.00000e+00],
[4.381141e-05,2.209878e-02,5.634949e-01]])
>>> dot(M,M) # Matrix multiplication
array([
[ 0.14061966,  0.25903369,  0.13984616],
[ 0.        ,  0.        ,  0.        ],
[ 0.00744346,  0.1158494 ,  0.56431808]])
```

Iterating over Array Elements

- In general, avoid iteration over elements
- Iterating is slow compared to a vector operation
- If you must, use the `for` loop
- In order to enable vectorization, ensure that user-written functions can work with vector inputs.
  - Use the `vectorize` function
  - Use the `any` or `all` function with arrays

Vectorize

Example

```
>>> def Theta(x):
...     """
...     Scalar implemenation of the
   Heaviside step function.
...     """
...     if x >= 0:
...         return 1
...     else:
...         return 0
...
>>> Theta(1.0)
1
>>> Theta(-1.0)
0
```

Vectorize

Without `vectorize` we would not be able to pass v to the function

Example

```
>>> v
array([1, 2, 3, 4])
>>> Tvec = vectorize(Theta)
>>> Tvec(v)
array([1, 1, 1, 1])
>>> Tvec(1.0)
array(1)
```

Arrays in conditions

Use the `any` or `all` functions associated with arrays

Example

```
>>> v
array([1, 2, 3, 4])
>>> (v > 3).any()
True
>>> (v > 3).all()
False
```

Section 4

SciPy

- SciPy framework built on top of the NumPy framework
- SciPy imports all the functions from the NumPy namespace
- Large number of scientific algorithms
  - Integration
  - Optimization
  - Linear Algebra
  - Sparse Eigenvalue Problems
  - Statistics
  - File I/O
  - Fourier Transforms
  - ... and many more

Lets look at some examples

Using any of these subpackages requires an explicit import

- Linear Algebra
- Optimization

Get system parameters

Example

```
>>> import sys
>>> sys.float_info
sys.float_info(max=1.7976931348623157e
    +308, max_exp=1024, max_10_exp=308, min
    =2.2250738585072014e-308, min_exp
    =-1021, min_10_exp=-307, dig=15,
    mant_dig=53, epsilon=2.220446049250313e
    -16, radix=2, rounds=1)
```

Linear Algebra

To solve an equation of the form $\mathbf{A}\,\mathbf{x} = \mathbf{b}$

Example

```
>>> from scipy import *
>>> from scipy import linalg
>>> A = array([[1,2,3], [4,5,6], [7,8,9]])
>>> b = array([1,2,3])
>>> x = linalg.solve(A, b)
array([-0.33333333, 0.66666667, 0. ])
>>> linalg.norm(dot(A, x) - b)
1.1102230246251565e-16
```

Linear Algebra - Inverse

Example

```
>>> A = random.rand(3,3)
>>> A
array([
 [ 0.24514116,  0.52587023,  0.18396222],
 [ 0.90742329,  0.16622943,  0.13673048],
 [ 0.09218907,  0.51841822,  0.5672206 ]])
>>> linalg.inv(A)
array([
[-0.13406351,  1.16228558, -0.23669318],
[ 2.87602299, -0.69932327, -0.76418374],
[-2.60678741,  0.45025145,  2.49988679]])
```

Linear Algebra - Eigenvalues and Eigenvector

Example

```
>>> evals, evecs = linalg.eig(A)
>>> evals
array(
[-0.46320383+0.j,  1.09877378+0.j,
   0.34302124+0.j])
>>> evecs
array([
[-0.49634545,  0.49550686, -0.20682981],
[ 0.79252573,  0.57731361, -0.35713951],
[-0.35432211,  0.64898532,  0.91086377]])
```

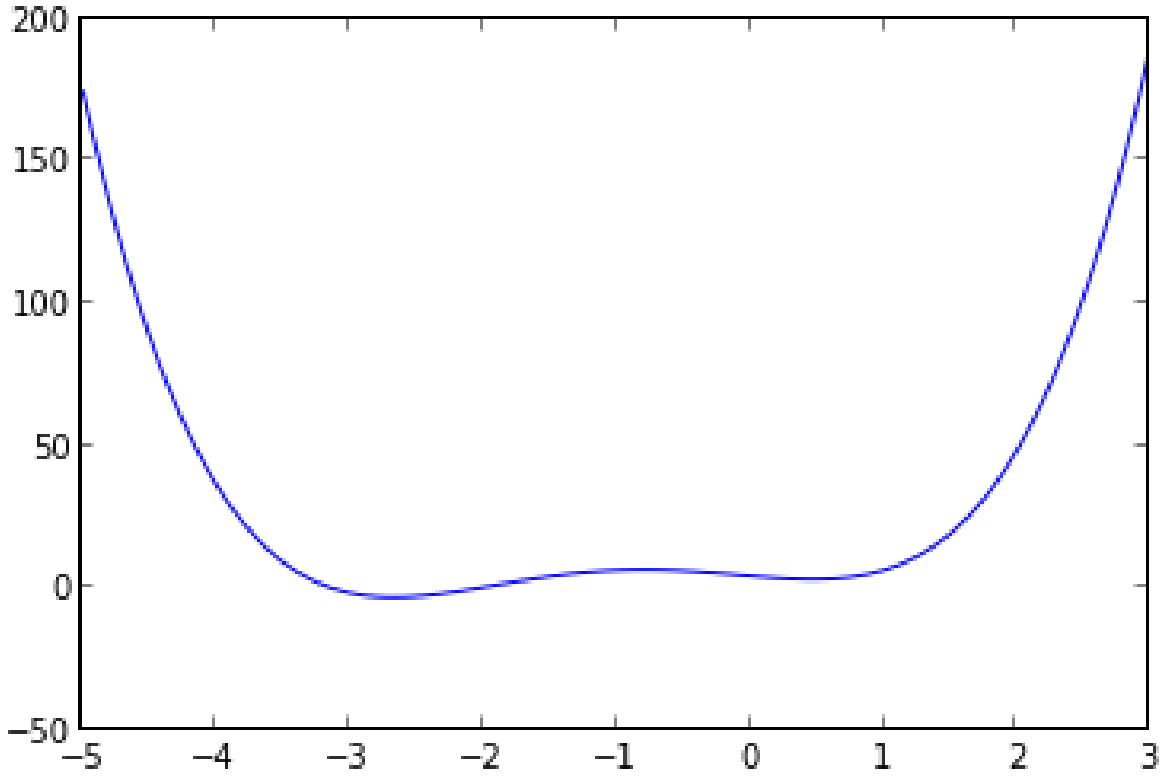
Optimization

Compute the minima of a single variable function

Example

```
>>> from scipy import optimize
>>> def f(x):
        return 4*x**3 + (x-2)**2 + x**4
```

Function f(x)



Optimization

Example

```
>>> x_min = optimize.fmin_bfgs(f, -2)
      Optimization terminated successfully.
      Current function value: -3.506641
      Iterations: 6
      Function evaluations: 30
      Gradient evaluations: 10
array([-2.67298167])
```

Section 5

Matplotlib

- Used for generating 2D and 3D scientific plots
- Support for LaTeX
- Fine-grained control over every aspect
- Many output file formats including PNG, PDF, SVG, EPS

Matplotlib - Customize matplotlibrc

- Configuration file 'matplotlibrc' used to customize almost every aspect of plotting
- On Linux, it looks in .config/matplotlib/matplotlibrc
- On other platforms, it looks in .matplotlib/matplotlibrc
- Use '`matplotlib.matplotlib_fname()`' to determine from where the current matplotlibrc is loaded
- Customization options can be found at `http://matplotlib.org/users/customizing.html`

Matplotlib

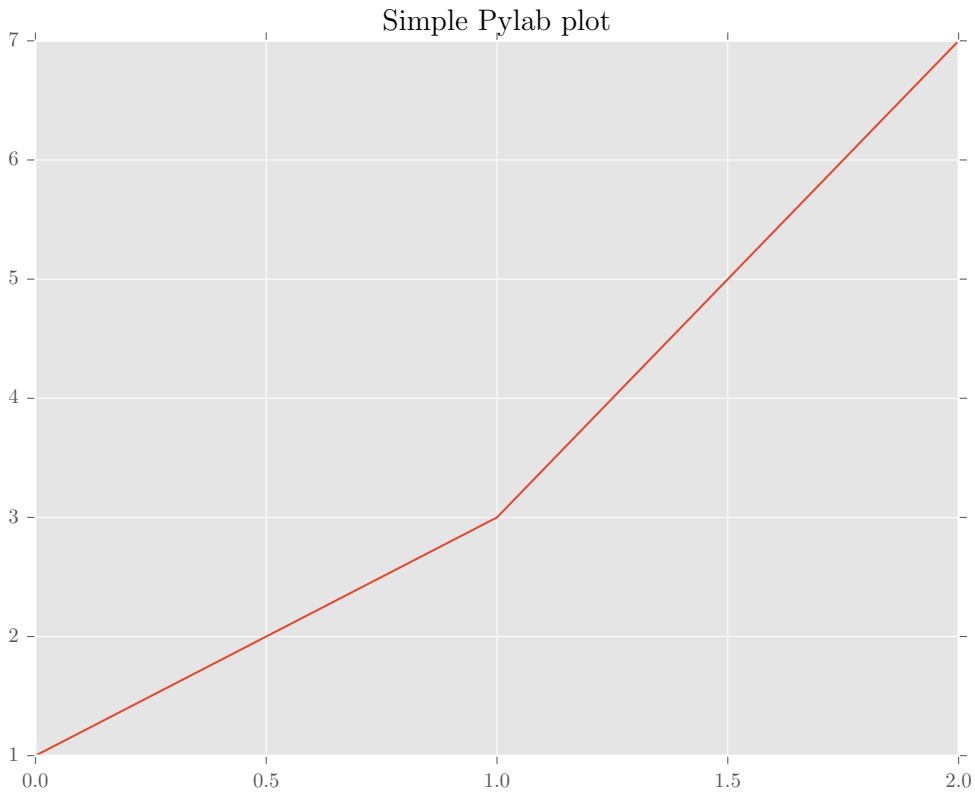
- Matplotlib is the entire library
- Pyplot - a module within Matplotlib that provides access to the underlying plotting library
- Pylab - a convenience module that combines the functionality of Pyplot with Numpy
- Pylab interface convenient for interactive plotting

Pylab

Example

```
>>> import pylab as pl
>>> pl.ioff()
>>> pl.isinteractive()
False
>>> x = [1,3,7]
>>> pl.plot(x)     # if interactive mode is
    off use show() after the plot command
[<matplotlib.lines.Line2D object at 0
   x10437a190>]
>>> pl.savefig('fig_test.pdf',dpi=600,
   format='pdf')
>>> pl.show()
```

Pylab

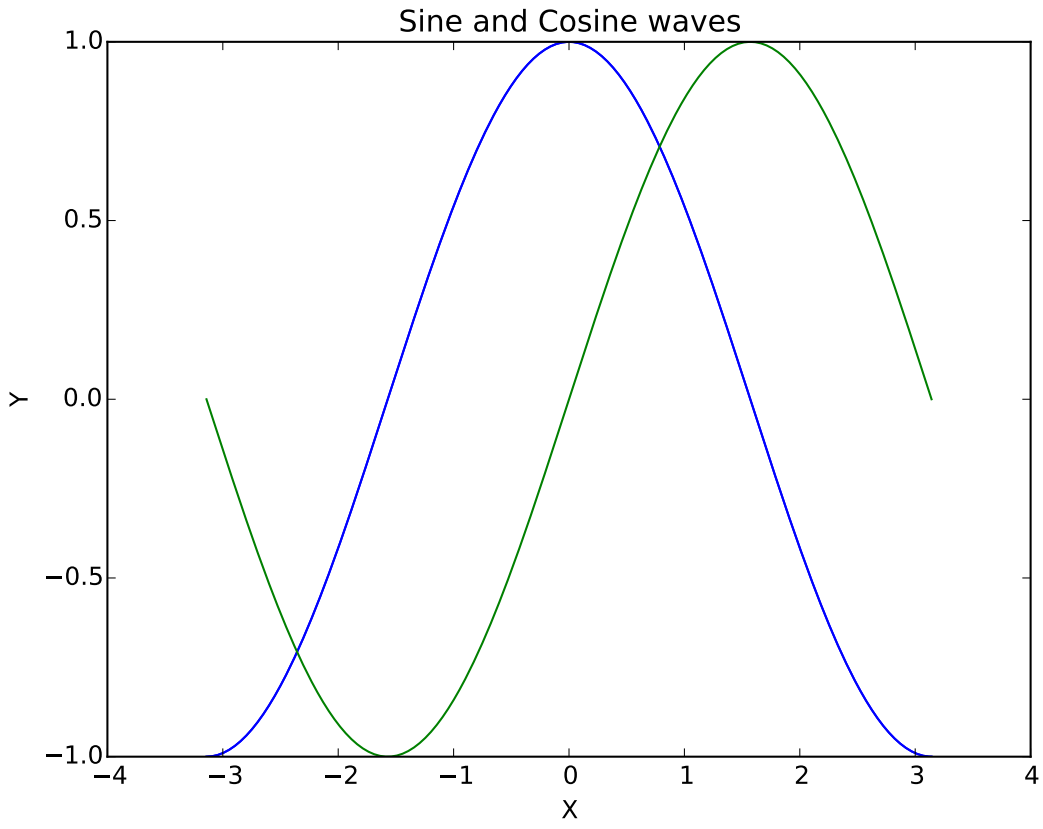


Pylab

Example

```
>>> X = np.linspace(-np.pi, np.pi, 256,
    endpoint=True)
>>> C, S = np.cos(X), np.sin(X)
# Plot cosine with a blue continuous line
    of width 1 (pixels)
>>> pl.plot(X, C, color="blue", linewidth
    =1.0, linestyle="-")
>>> pl.xlabel("X") ; pl.ylabel("Y")
>>> pl.title("Sine and Cosine waves")
# Plot sine with a green continuous line
    of width 1 (pixels)
>>> pl.plot(X, S, color="green", linewidth
    =1.0, linestyle="-")
>>> pl.show()
```

Pylab

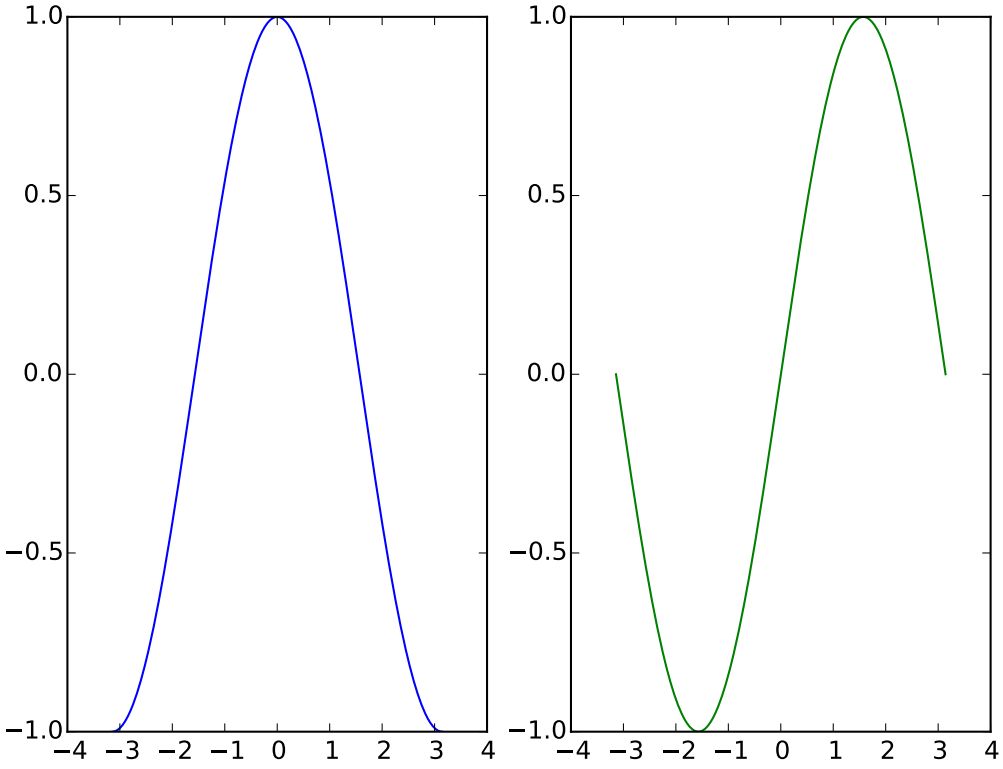


Pylab - subplots

Example

```
>>> pl.figure(figsize=(8, 6), dpi=80)
>>> pl.subplot(1, 2, 1)
>>> C, S = np.cos(X), np.sin(X)
>>> pl.plot(X, C, color="blue", linewidth
    =1.0, linestyle="-")
>>> pl.subplot(1, 2, 2)
>>> pl.plot(X, S, color="green", linewidth
    =1.0, linestyle="-")
>>> pl.show()
```

Pylab - subplots



Pylab - xlim, ylim

Example

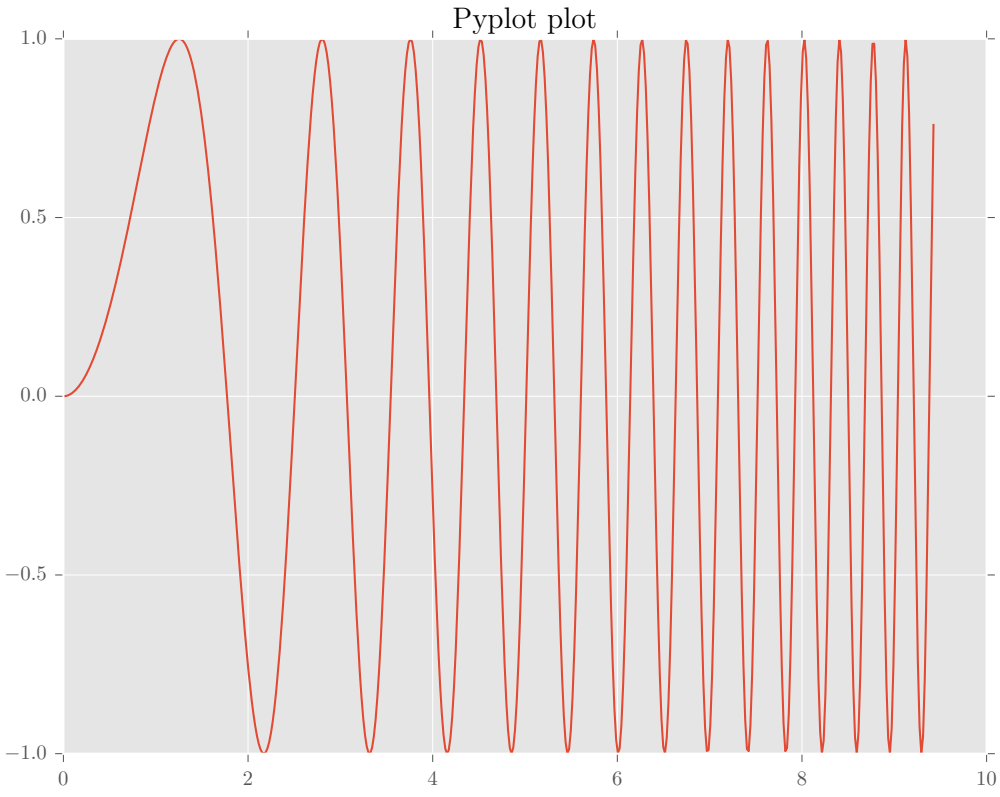
```
...
...
...
# Set x limits
>>> pl.xlim(-4.0, 4.0)
>>> pl.xticks(np.linspace(-4, 4, 9,
    endpoint=True))
# Set y limits
>>> pl.ylim(-1.0, 1.0)
# Set y ticks
>>> pl.yticks(np.linspace(-1, 1, 5,
    endpoint=True))
>>> pl.show()
```

Pyplot

Example

```
>>>import matplotlib.pyplot as plt
>>>plt.isinteractive()
False
>>>x = np.linspace(0, 3*np.pi, 500)
>>>plt.plot(x, np.sin(x**2))
[<matplotlib.lines.Line2D object at 0
   x104bf2b10>]
>>>plt.title('Pyplot plot')
<matplotlib.text.Text object at 0
   x104be4450>
>>>savefig('fig_test_pyplot.pdf',dpi=600,
   format='pdf')
>>>plt.show()
```

Pyplot

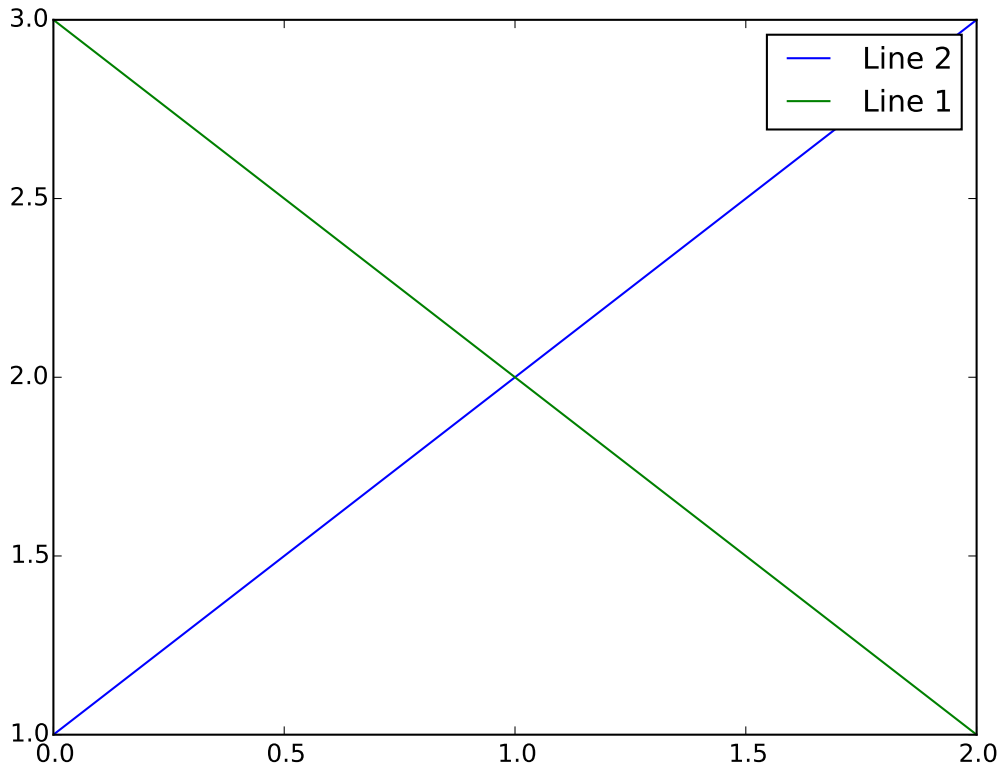


Pyplot - legend

Example

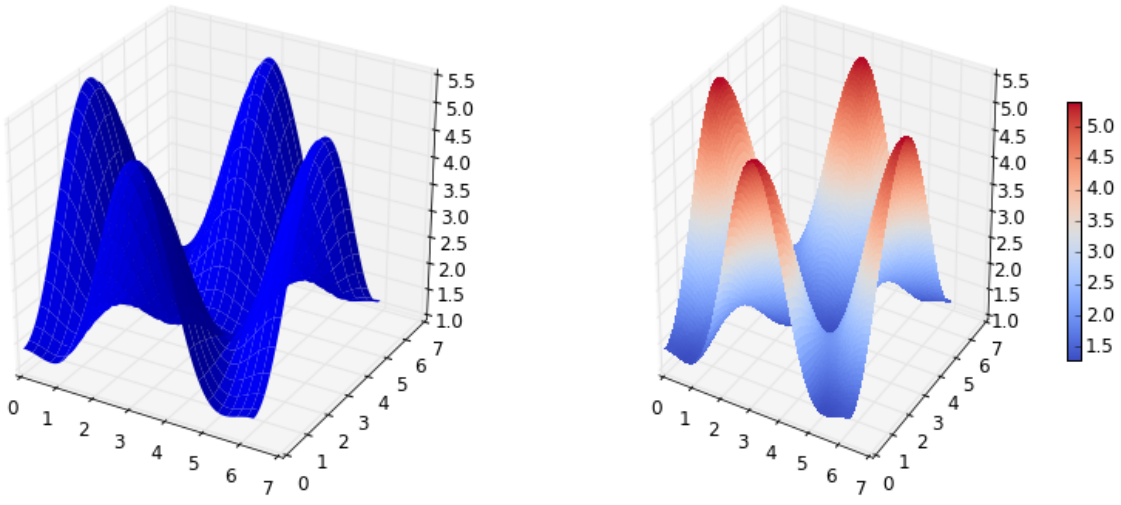
```
>>> import matplotlib.pyplot as plt
>>> line_up, = plt.plot([1,2,3], label='
    Line 2')
>>> line_down, = plt.plot([3,2,1], label='
    Line 1')
>>> plt.legend(handles=[line_up, line_down
    ])
<matplotlib.legend.Legend at 0x1084cc950>
>>> plt.show()
```

Pyplot - legend



Pyplot - 3D plots

Surface plots



Visit `http://matplotlib.org/gallery.html` for a gallery of plots produced by Matplotlib

Section 6

Conclusion

- Python used extensively by the educational and scientific community
- Used as both a scripting and prototyping tool
- Plenty of libraries out there
- Extensively documented !

Questions

Thank you for attending !