AMS 595: Fundamentals of Computing: Part II

Lecture 1: Overview of Python

Xiangmin Jiao

Stony Brook University

Outline

Why Use either Python or MATLAB?

Similarities

- Very high-level, interpreted programming languages
  - Dynamic typing
  - Automatic memory management (garbage collection)
  - Code can be modified/generated on the fly
- Can be used for scripting or complicated modules/applications
- Great for prototyping
- Have many packages available, with different focuses

Why Use Python instead of MATLAB, and Vice Versa?

Differences

- The most fundamental difference: MATLAB uses pass-by-value (copy-on-write), whereas Python uses pass-by-reference
- MATLAB is better for numerical computations at the language level (although Python packages NumPy/SciPy can perform similar tasks)
- Python is object-oriented by design, often used for systems programming, text processing, web programming, etc.
- Python is often used as "gluing" language to link packages written in other languages (although MATLAB can also call C/C++, Fortran, Java and vice versa)
- Python is open-source and more portable, but MATLAB is proprietary (Octave is open-source and MATLAB-compatible at language level)

Running Python vs. MATLAB

- MATLAB has built-in GUI; matlab command starts GUI by default
- In contrast, Python itself starts in shell mode (text mode)
  - python: basic python shell (preinstalled in Linux/Mac)
  - ipython: powerful interactive shell with help, auto completion, etc.
- External GUI environments are available for Python, such as
  - Jupyter Notebook: a web app that allows you to create and share documents with live code, equations, visualizations, commentary, etc.
  - Spyder: Scientific PYthon Development EnviRonment, with advanced editing, interactive testing, debugging and introspection features, and integrated support for IPython, NumPy (linear algebra), SciPy (signal and image processing) or matplotlib (interactive 2D/3D plotting).
  - PyCharm: Popular for professional developers in industry

Modularity and Namespace

- In MATLAB, all functions in path can be used directly
- In Python, except for built-in functions, functions and variables are in modules, and you must import a module before you can use it
- Try it out:
  - Start octave at http://octave-online.net, and run:
    - sin(1)
    - cosd(45)
    - path
  - Start a python shell and run:
    - **import** math; math.sin(1)
    - math.cos(math.radians(45))
    - **import** sys; sys.path
- Python can also import function names into current name space; e.g.:
  - **from** math **import** sin; sin(1)
- Q: What are the pros and cons of the two approaches?
  - Convenience vs. name conflicts

Outline

Python 2.x vs. Python 3.x

- Mostly about cleaning up the language and making things consistent
  - e.g. print is a statement in python 2.x but a function in 3.x
- Some trivial differences
  - .pyc files are now stored in a `__pycache__` directory
- Some gotchas, for example
  - 1/2 will give different results between python 2 and 3
- It is possible to write code that works with both python 2 and 3, often by importing from `__future__`. For example,
  - In python 2.6+, use statement
    `from __future__ import print_function`
    and then use the new print() style
- See https://docs.python.org/3/whatsnew/3.0.html and https://wiki.python.org/moin/Python2orPython3
- **We will be using Python 3.x for this class**

Installing and Running IPython

- Installation
  - Install IPython, Jupyter Notebook and Spyder via Anaconda3
  - Add <anaconda_root>/bin to your path (e.g., $HOME/anaconda/bin)
- IPython Shell: a powerful interactive shell for Python
  - type `ipython` at prompt to start
  - type `%quickref` to see an overview
  - Build-in help with ? after function or variable name. For example:
    - `math.sin?` (after import math)
    - `x?` (after assigning x=1)
  - Magics (`%lsmagic` lists all the magic functions), e.g.
    - import numpy as np
    - **%timeit** np.linalg.eigvals(np.random.rand(100,100))
  - Tab completion
  - Run system commands (prefix with !)
  - Last 3 output objects are referred to as `_`, `__`, `___`

Jupyter Notebooks

- A web-based environment that combines code and output, plots, plain text/stylized headings, LaTeX (later versions), etc.
  - Notebooks can be saved and shared
  - Viewable on the web via: http://nbviewer.ipython.org/
  - Provides a complete view of your entire workflow
- Jupyter Notebook (previously known as IPython Notebook)
  - type `jupyter notebook` (or `jupyter-notebook`) at prompt
  - We will provide notebooks for most lectures

Topics to Be Covered

- Data types, data structures, and exceptions
- Control flow and functions
- File I/O and regular expressions
- Scripts and modules; debugging and testing
- Array computation and curve plotting
- Numerical and symbolic computations
- Data analysis with Pandas
- Object-oriented programming
- Building Python applications and language interoperability

Outline

Basic Data Types

- Python has basic data types as most other languages: integer, floating point, complex, strings
- Built-in numbers
  - `int` has variable length (in Python 2.x, `long` for variable length)
  - `float` is double precision
  - complex uses 'j' for imaginary parts
  - More integer and float types are provided in NumPy
- Strings
  - Use triple quotes for multiline strings
  - Escape characters and raw strings
  - '+' for concatenation and '*' for replication
  - 0-based indexing; negative indexing; slicing excludes the end
- All types are classes
- Variables are case sensitive
- Demo: Jupyter notebook on data types