USING PYTHON, AN INTERACTIVE OPEN-SOURCE PROGRAMMING LANGUAGE, FOR PLANETARY DATA PROCESSING.

J. Laura[1,2], T. M. Hare[2], and L. R. Gaddis[2], [1]School of Geographical Sciences and Urban Planning, Arizona State University, Tempe, AZ; [2]Astrogeology Science Center, U.S. Geological Survey, Flagstaff, AZ. (jlaura@asu.edu).

Introduction: Python is an open-source programming language ideally suited to perform planetary data analysis, processing, and exploration. Python is epitomized by rapid code development, an iterative workflow, and source readability. Python offers a variety of freely available libraries to generate interactive visualizations, perform complex data analysis, and collaboratively share repeatable results with colleagues. Here we describe the use of Python as a language for developers and describe the key features of Python for planetary researchers. We then discuss three use cases for which Python has provided an ideal development and deployment environment: as a language used to develop a stand-alone image analysis suite, as an interactive environment for data exploration, and as a development platform for small scale data visualization and large scale model derivation.

Python is Ideal for Developers: Python provides a rapid, interactive development environment that does not require code compiling. This environment fosters a simple workflow (e.g., implement, test, refactor, utilize) that facilitates efficient code prototyping and delivery. For example, code can be rapidly developed and delivered with loose requirement specifications, then iteratively refactored with input from science users as needs are refined. Python is easily learned by new users because of two development principles: (1) code should be human readable and (2) code should be explicit in its implementation [1].

Python offers a variety of open-source code libraries that can greatly simplify development. NumPy (Numerical Python) and SciPy (Scientific Python) provide a suite of tools for working with raster (array) data suitable for image analysis. GDAL (Geospatial Data Abstraction Library) provides access to over 120 image data formats, including PDS, ISIS2/3, FITS, ENVI, Jpeg2000, Png and Tiff [2]. MatPlotLib, Pandas, and Basemap libraries provide standard and cartographic data visualization tools. Taken together, these libraries integrate into a highly effective processing suite for scientific data exploration, visualization, and analysis.

Python's relatively slow dynamic code execution speed compared to traditional compiled languages is an oft-cited shortcoming [3]. However, Python provides multiprocessing options that allow for rapid implementation of high level multi-core code. Additionally, many Python libraries are wrappers for underlying C code, which offers improvements in speed. NumPy, for example, is optimized for high-speed, vectorized computation using well known C libraries.

Finally, the ability to rapidly develop code in Python means that it is an ideal sandbox language that supports code and algorithm testing prior to implementation in other software packages. For example, image analysis algorithms implemented in Python can be readily tested for validity and usability prior to being ported to C++ for integration into a package like the USGS ISIS3 [4] or IDL for ENVI.

Science Team Support. Python is pre-installed or deployable using either *apt-get* or binaries on all modern platforms. Installation of multiple additional modules can be challenging, but is simplified by the use of EPD (Enthought Python Distribution). EPD is free for academic users and provides an all-in-one deployment that includes all of the modules described above. End-user support using EPD requires only a simple binary installation for all platforms and the briefest introduction to the command line. Alternatively, Python code can be wrapped into a Windows executable or OSX application and shipped with dependencies included.

Python is Ideal for Scientists: Beyond rapid development and associated lower costs, Python provides an interactive environment for data exploration, analysis, and visualization. For example, IPython allows for 'open, collaborative, reproducible scientific computing' through the use of interactive visualization, a web notebook to store 'code, text, mathematical expressions, inline plots' and '[e]asy to use, high performance tools for parallel computing' [5].

Image Analysis: PyStretch is a processing and analysis tool built entirely in Python. At its core, PyStretch provides a means to manipulate images larger than available system memory (e.g., 4 GB) while maintaining essential geospatial data properties such as projection, cell size, and image coordinate system [6, 7]. To perform this processing, images are first ingested using GDAL and converted to arrays. Each array is then segmented twice, first into subsections that fit into the system's available memory (GB of RAM), and then again into smaller chunks in preparation for multiprocessing. This transformation from on-disk geospatial image to an array stored in RAM occurs completely in shared memory space and does not require any image duplication. The implications are that processing times are reduced substantially as disk I/O is slower than in-memory processing. Once distributed, image subsections are processed using a variety of included algorithms (e.g., linear and non-linear stretches or filters) and written to disk. In addition to pixel manipulation, data products can be scaled, reformatted, and altered in data type (e.g., 32-bit to 16-bit).

PyStretch has been tested internally on a dual-core, 4GB RAM, MacBook Pro processing a 522MB 8-Bit GeoTiff image in 0:15 seconds, a 1GB 32-bit GeoTiff in 1:02 minutes, and an 8.88GB Compressed GeoTiff (16.99GB uncompressed) in 12:13 minutes. These times are expected to improve substantially with the next rollout of PyStretch. *This use case example highlights the ability for Python to leverage outside libraries to perform planetary data analysis, the use of multi-core processing to facilitate analysis of large data sets, and the ability to perform complex computation at speeds comparable to compiled software solutions.*

HyperSpectral Image Analysis: The science team of the Mars Reconnaissance Orbiter (MRO) CRISM hyperspectral instrument has published more than 40 algorithms to create derived spectral parameter products [e.g., 8]. Many of these are sufficiently complex that a command-line implementation is most suitable. Others are ideal for interactive data exploration. For example, ICER2 is a simple band ratio of the CRISM 2350µm band and the 2600µm band to highlight CO_2 ice on the Mars surface. Although the CRISM team uses Exelis' ENVI application, the open-source IPython suite provides a simple, cost-effective solution for deriving these CRISM spectral data products.

IPython offers an interactive Python shell and an integrated command line shell, so one can access a text editor (vim, nano), GDAL command line tools, and all available Python modules. To perform the ICER2 band ratio, users can simply download their desired image, determine the band numbers that represent 2350µm and 2600µm using the appropriate wavelength.tab lookup table, read each band as a NumPy array, derive the ratio using standard mathematical notation (Array_1/Array_2), and write the output to disk using GDAL's Python bindings. A user needing only to visualize the results could plot the data using MatPlotLib without having to write the results to disk. *This use case example highlights the suitability of Python to operate as an interactive data analysis environment that provides usage familiar to scientists in an open-source package.*

Eruption Modeling and Simulation: Python also provides a tool for interactively modeling and visualizing geologic processes such as volcanic eruptions. For example, the simple ballistic eruption model for emplacement of the Orientale annular pyroclastic eruption [9] can be modeled and viewed using Python in 3D [e.g., 10] as pyroclasts are emplaced randomly from the observed fissure vent onto a flat-surface mosaic and then an elevation model of the lunar surface [11, 12]. First, a basic model ejecting particulates from a single point onto a planar, Equirectangular projected, surface was coded and tested. Users define ejection angle and velocity ranges along with the number of sequentially ejected points. Existing Python libraries (MatplotLib, Basemap, & NumPy) were used to rapidly prototype a geospatial visualization that depicts point particulate landing sites. A second iteration of the code introduced a linear ejection feature, randomly selecting a potential ejection point, as well as topographic impact checking, using GDAL to extract a topographic profile from the WAC DTM. This iteration also included a point-density visualization, with user definable grid size. A final iteration provided the means to write the model output to a shapefile, with each point tagged with a temporal identifier, using either a single core, or all available cores (multiprocessing). This allows for rapid 3D visualization of hundreds of thousands of particulate depositions. The ability for the science team to interactively test low numbers of particulate deposition over the WAC basemap mosaic, iteratively test ejection angle and velocity combinations, and finally run large scale models output as a shapefile was particularly beneficial. *This use case example highlights the potential to iteratively design and develop interactive visualizations for the exploration of planetary surface processes through the integration of existing, tested, Python libraries as well as the potential to develop low overhead visualizations that promote an integrated, single application, science workflow.*

Conclusion: Python offers a development and data analysis platform ideal for scientific analysis of planetary data. The potential unique nature of planetary data is not insurmountable and three usage examples provide an overview of the potential benefits achievable using Python.

References: [1] van Rossum (2001), Style Guide for Python Code, http://www.python.org/dev/peps/pep-0008/. [2] Hare et al. (2008), LPS XXXIX, abs. 2536. [3] van Rossum (2012), Interview, see http://tinyurl.com/7jl3jev. [4] Anderson et al. (2004), LPSC XXXV, abs. 2039. [5] Pérez and Granger (2007), Comp. Sci. Engin., v9, pp. 21-29. [6] http://pypi.python.org/pypi/PyStretch/. [7] Laura (2012), NASA Workshop for Data Users and Developers, June 2012. [8] Pelkey et al. (2007), JGR Vol. 112. [9] Head et. al. (2002), JGR Vol. 107. [10] Gaddis et al., this volume. [11] Speyerer et al. (2011), LPS 42, #2387. [12] Scholten et al. (2012), JGR, v. 117, E3, DOI: 10.1029/2011JE003926.