VS

- Developed by programmers
- Good style and clear rules enforced
- Few functions
- Numerical packages are cutting edge
- Statistical packages are still relatively young
- Ecologists not familiar with

- Developed by stat gurus
- Style not well defined or enforced
- Lots of functions
- Numerical operations not as powerful
- Statistical packages are top notch
- Ecologists familiar with
Our Goals

• Call python and R scripts from the shell

• Evoke the command line within R and python

• Use Python modules to interactively call R
The Shell
aka the terminal or command prompt

• Provides a textual way to interact with your OS – control files, processes, and networking

• We can use the shell to interact directly with R and python

• Examples

  $ python my_python_script.py

  $ Rscript my_r_script.R
Communicating with the terminal

• In Python
  ```python
  >>> import os
  >>> os.system("python my_python_script.py")
  >>> # alternatively call an R script
  >>> os.system("Rscript my_r_script.R")
  ```

• In R
  ```r
  > system("python my_r_script.r")
  > # alternatively call a python script
  > system("python my_python_script.py")
  ```
Time to Try It out

• Create a simple python script that prints anything to the console
• From the shell call your script
• From the python interpreter call your script using
  >>> import os
  >>> os.system("python my_python_script.py")
Python modules to link python and R

- RSPython
  - last development in 2005
  - allows bidirectional interactive sessions
- pypeR
  - no recent (i.e. last year) development activity
  - uses pipes to establish interactive R sessions
- pyRserve
  - in beta but stable
  - uses Rserve to establish interactive R sessions
- Rpy/Rpy2
  - most popular module for interfacing with R
  - python to R interactive sessions
pyRserve

• Connects to an R process via Rserve
• Each R instance is like connecting to a server
• Pros
  – Can run on a remote machine
  – Allows easy parallelization of R processes
  – Pythonic style
  – Plays nice with numpy
• Cons
  – Installing Rserve can be challenging even in Linux
pyRserve examples

```python
>>> conn = pyRserve.connect()
>>> conn.r("3 + 4")
7.0
>>> conn.r("mean(c(3, 4, 5))")
4
>>> conn.r("a = 3")
# or alternatively set a with an attribute
>>> conn.r.a = 3
>>> print conn.r.a
3
```
Rpy/Rpy2

- Rpy is older and no longer being developed
- Rpy2 adds greater capabilities and object classes
- Rpy2 is the backbone of Rmagic in ipython

Pros
- Rpy & Rpy2 are popular - there is a user group to query when you have trouble
- Play nice with numpy
- Pythonic style
- Rpy and Rpy2 are easy to install in Linux

Cons
- In Windows, it is difficult to get Rpy2 installed; however, Rpy is straightforward to install.
Rpy/Rpy2 sub-packages

- **rpy2.rinterface**
  - Low-level interface to R, when speed and flexibility matter most. Close to R’s C-level API.

- **rpy2.robjects**
  - High-level interface, when ease-of-use matters most. Should be the right pick for casual and general use. Based on the previous one.

- **rpy2.interactive**
  - High-level interface, with an eye for interactive work. Largely based on rpy2.robjects.

- **rpy2.rpy_classic**
  - High-level interface similar to the one in RPy-1.x. This is provided for compatibility reasons, as well as to facilitate the migration to RPy2.

- **rpy2.rlike**
  - Data structures and functions to mimic some of R’s features and specificities in pure Python (no embedded R process).
Time for an Rpy2 Demo