Lecture 08 - The why and how of scripting

What's scripting?

- automating a computational process using a programming language

- often this occurs via a `read-eval-print` loop (REPL)
  - the *read* function accepts and parses a submitted code expression
  - the *eval* function identifies the called functions and implements them with the supplied arguments
  - the *print* function passes what is returned from the function calls to the user
  - the system than returns to the *read* state creating a loop

- this process can occur at a prompt, but can also be implemented on a text file containing code, which we often refer to as a **script**

Why script?

Why script?

- efficiency
- reproducibility
- get a job
- impress your family and friends
- preserve your sanity

Scripting is the future - Readings. . .

- Big data is everywhere
- Transition between 2006 and 2016 in needs/priorities
- Needs for training not being met
- Training is perceived to be a requirement for future success

Characteristics of programming languages

- portability
- computational efficiency
- ease of use
- specificity

Compiled vs. interpreted languages

- **Compiled** - source code is translated to computer-understandable instructions once prior to its use. The compiling process is specific to a given computer based on its hardware, operating system, etc. Examples include C, C++, and sometimes Java.

- **Interpreted** - human-readable code is read and evaluated by an interpreter each time it is run. However, the interpreter itself is a compiled program. Examples include Python, R, and Perl.

Characteristics of programming languages

- portability
- computational efficiency
- ease of use
- specificity

Compiled vs. interpreted languages

What is `bash`?

Scripting languages

What are some scripting languages you've heard of before?

Where do they fall in the programming language space?

When to use `bash` and when to use `R`/`Python`?

I find there's no reason to use Bash/Shell if you have Ruby/Python installed ... Why do people use Bash nowadays? Because it's a terrible, hard-to-break habit.

Given a problem that both can handle, you'll want to use the one you're most comfortable with. Ultimately, there are a lot of small details, and only experience can teach you to see them.

When to use `bash` and when to use `R`/`Python`?

I use **bash** when my primary focus is on **file handling**. This could include moving, copying, and renaming files.

I use Perl and **python** when my primary focus is on reading data from files, **processing that data** in some way, and writing output to files.

If I find myself using the subprocess module too extensively, I consider writing the script in bash. On the other hand, I sometimes start adding so much functionality to a bash script that eventually it makes more sense to rewrite it in python rather than deal with bash's limited (by comparison) support for variable scoping, functions, data structures, etc.

How this section of class will work. . .

As I said the first day or two of class we are going to focus on Python *or* R.

Lectures will focus on theory of both and tutorials will focus on code-specific application

In general R code will appear in the lower left for examples.

Python code will be here in the lower right.

Python vs. R

The differences between these two languages, other than their syntax, have diminished dramatically over the last 2-5 years. Both can run in *interactive* or *batch* mode, and have a broad base of packages for augmenting their functionality.

- `R` evolved in the early 1990's from `S`, which was developed at AT&T's Bell Labs
- Python was developed by Guido von Rossum in the early 1990's as a "hobby project"

Python vs. R

The "major" remaining differences include:

- `R` relies heavily on packages for added functionality and therefore hundreds or thousands of packages exist
- `R` grew out of the statistics arena and therefore still has better abilities for complex statistical modeling and math, especially in the base package

- `Python` has more human readable code
- `Python` is faster

Open source and a community of contributors

Both R and Python have broad user communities that develop packages and contribute to user forums.

- R uses the Comprehensive R Archive Network (CRAN)
- Python uses the Python Package Index (PyPI)

Installation

Both languages are fairly easy to install thanks to development projects that streamline the process of managing packages and working in an interactive manner.

- Rstudio is a relatively new interactive development environment (IDE) for `R`
- It allows for package management and adds other functionality like interacting with `Git` and `Github`

- An interactive version of `Python` (`IPython`) operates similarily to `R`
- Anaconda is a "Python Data Science Platform" that manages packages and provides an easy to install version of `Python` and `IPython`
- Use version 2.7!!!

Topics to be covered

- variables, input/output, data structures
- documentation: markdown and knitr
- Control flow
  - making decisions
  - looping
- Regular expressions and string manipulation
- Package repositories
- Making graphics
- Applications