# Scripting Languages

# Overview

- What are Scripting languages?
- Characteristics
- Some examples
- BASH 101
- Python introduction (by examples)

# What are Scripting Languages?

- Hard to say
- Glue languages: put things together
- Coordination languages
- Programming languages such as C, C++ or Java are too “clunky”
- If system performance is not a priority, then C/C++ is not necessary
- If safety and modularity is not necessary, why use Java?
- Usually just want to write something small, quickly, and for a very specific purpose: scripting

# General Characteristics of Scripting Languages

- Weakly typed:
  - Can easily convert from string to numbers
  - Use of variants
- Dynamically typed: heavy reliance of type inference, type compatibility
- Interpreted languages (with likely some capability for Just-In-Time/JIT compilation)
- Implicit variable declaration: first use instantiates the variable
- Relatively easy and simple I/O mechanisms
- Usually very compact syntax, which translates to little bloat
- High-level data types built-in (lists, sets, maps, strings, arrays, REGULAR EXPRESSIONS)

# Some Scripting Languages

- Sh / C-Shell / Bash
- Python
- Perl
- PHP
- Javascript (eeeewwww)
- MatLab, R
- Visual Basic Scripting (you can program your own macros in Excel)

# PHP

- Interpreter
- Allows to generate “dynamic web pages”
- Static web page is one with .html extension
- Works in tandem with a web server (e.g. Apache)

```
<!DOCTYPE html>
<html>
<body>
<h1>Developer News</h1>
<?php echo ”A PHP Example"; ?>
</body>
</html>
```

```
<?php
// Assign the value "Hello!" to the variable "greeting"
$greeting = "Hello!";
// Assign the value 8 to the variable "month"
$month = 8;
// Assign the value 2019 to the variable "year"
$year = 2019;
?>
```

# Shell / Bash

- Shell: command line interface available in Unix/Linux systems (also Mac OSX)
- BASH (Bourne Again Shell)
- (I prefer BASH)
- Available in command line / terminal environments
- Close interaction with Operating Systems
- File management:
  - Search for files in the file system
  - List files in directory
  - Create, rename, delete and move files
- Usual control-flow constructs:
  - Iterate
  - Decision control

# Bash 101

- Open a Linux / Mac OSX command line and try
- Comments: command line interpreter ignores all characters in a single line after finding a ‘#’
- Listing files:

```
ls -l # list all file in wide format
ls -lh # same as above, and print file size in human legible form
ls -lhr # same as above and show most recent below
```

- For more, do:

```
man ls
```

# Bash 101

Try this:

```
x=1
if [ $x -eq 1 ]; then
  echo ”x”
else
  echo “$x”
fi
```

- The above will print “x”
- Now set x to 2 (x=2) and re-run the rest, you will get 2 as output
- Always leave at least one space between left ‘[‘ and right ‘]’
- Other comparison operators: -ne, -lt, -leq, -gt, -geq
- For more options do: `man test`

# Bash 101

Now try this:

```
touch emptyfile.txt
ls –lh emptyfile.txt
if [ -s emptyfile.txt ]; then
  echo “File exists and has size GT zero”
else
  echo “File doesn’t exist or size not GT zero”
```

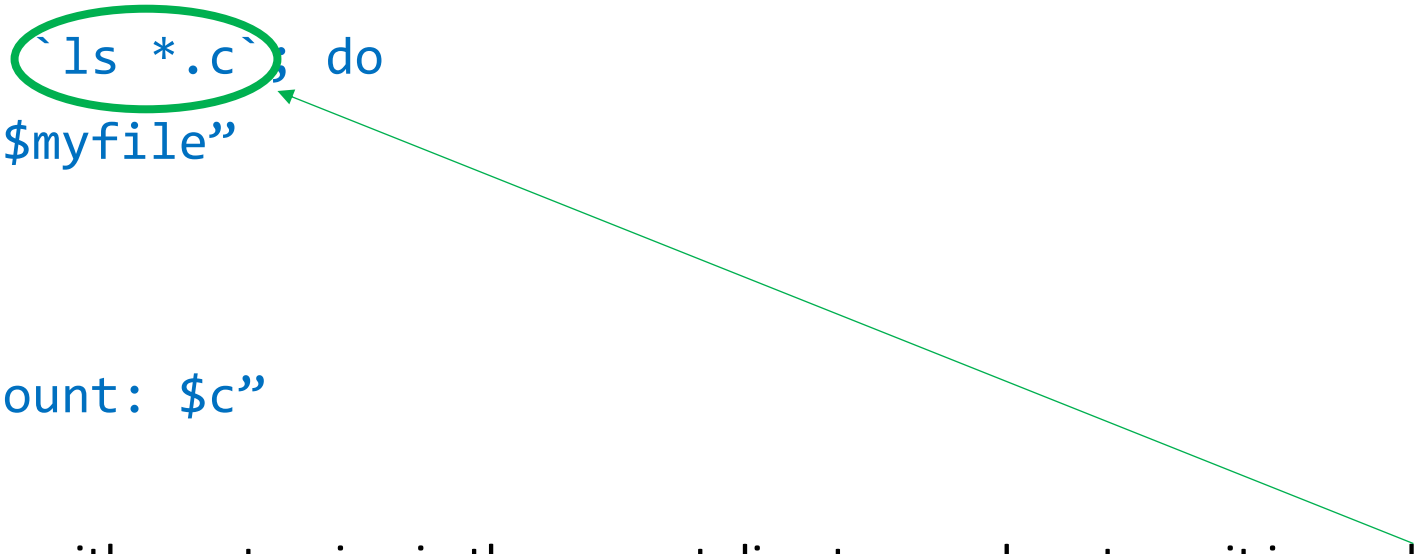
The above:

- Creates a file with size 0
- Shows that it exists (but has size 0 bytes)
- The IF uses the -s option for file test: must exist and have size 0

# Bash 101

Now try this:

```
c=1
for myfile in `ls *.c`; do
  echo “$c : $myfile”
  c=$(($c+1))
done
echo “Final count: $c”
```



- The above:
- Collects the files with .c extension in the current directory and captures it in a sub-shell
- Iterates over the files collected, increments the counter
- For each file, shows a number and its name
- Print the final count of files collected

# Bash 101

```
for myfile in `ls *.c`; do
  has=`grep switch $myfile`;
  if [ ! "x$has" == "x" ]; then
    echo "File $myfile has a switch";
  else
    echo "File $myfile not matched";
  fi;
done
```

- The above program uses grep, a regular expression matcher/search utility
- It will return a non-empty string if a file matches the regular expression provided, which is “switch”
- The IF, checks that variable $has is not empty

# Bash 101

My personal favorites:

- sed (utility for regular expressions):

  echo “programming languages” | sed –e ‘s/a/A/g’
- top #list processes running, quit with ‘q’
- ps ax #somewhat similar to top, different format
- head -10 filename #shows first 10 lines of a file
- cat filename # shows contents of file
- seq N # try with N=5
- grep switch *.c # search for ‘switch’ in all *.c
- wc # “word count”, but counts many things, try with –w and –l

  ls –l | wc -l
- tail -10 filename #shows last 10 lines of a file
- find (finds a file in some directory):

  find .  -name grammar.y # if you know the exact name

  find . | grep gram # try it

# Bash 101

Read about:

- Redirections: > >> < |
- Other commands:
  - mv from to
  - cp from to
  - rm afile
  - which anycommandyouknow

NEVER DO: rm *

# Python

BASH is cool, but if you need to compute a few statistics from files, or to plot, Python is much much better

So I personally like Python for two things:

- Plotting figures
- Searching files with strings and regular expressions: import re

Some Python features:

- Syntax and block scoping: indentation
- Has Lambda support
- Interpreted and cross platform
- Automatic garbage collection
- Support object oriented features
- C/C++ bindings (i.e. you can call C/C++ code from it)

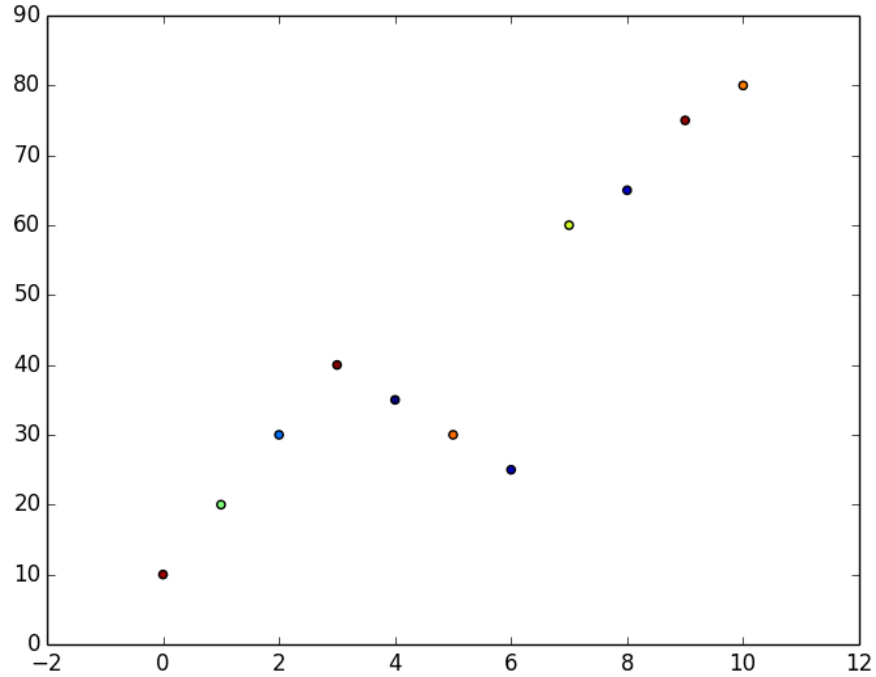
# Python Example 1

```
#!/usr/bin/python

import numpy as np
import matplotlib.pyplot as plt

y = [10,20,30,40,35,30,25,60,65,75,80]
N = len(y)
x = range(len(y))
colors = np.random.rand(N)

plt.scatter(x, y, c=colors)
plt.show()
```



# Python Example 2

```
#!/usr/bin/python

import os
import sys

if ( len(sys.argv) != 2):
  print ("Usage: ./show-file.py <filename>")
  sys.exit (1)

filename=sys.argv[1]
print ("Filename received: {}".format (filename))

ff = open (filename, "r")
for line in ff.readlines ():
  line = line.strip ()
  print (line)

ff.close ()
```

# Python Example 3

```
#!/usr/bin/python
import os
import sys

def myavg (line):
  parts = line.split (":")
  s = 0
  for pp in parts:
    v = float (pp)
    s += v
  s = s / len(parts)
  return s

if ( len(sys.argv) != 2):
  print ("Usage: ./my-average.py <filename>")
  sys.exit (1)
```

```
filename=sys.argv[1]
print ("Filename received: {}".format
(filename))

ff = open (filename, "r")
for line in ff.readlines ():
  line = line.strip ()
  a = myavg (line)
  print ("{} ==> {}".format (line,a))

ff.close ()
```

```
1:2:3
3:4:6:7
3:4
```

```
1:2:3 ==> 2.0
3:4:6:7 ==> 5.0
3:4 ==> 3.5
```