Day 8: System Interaction

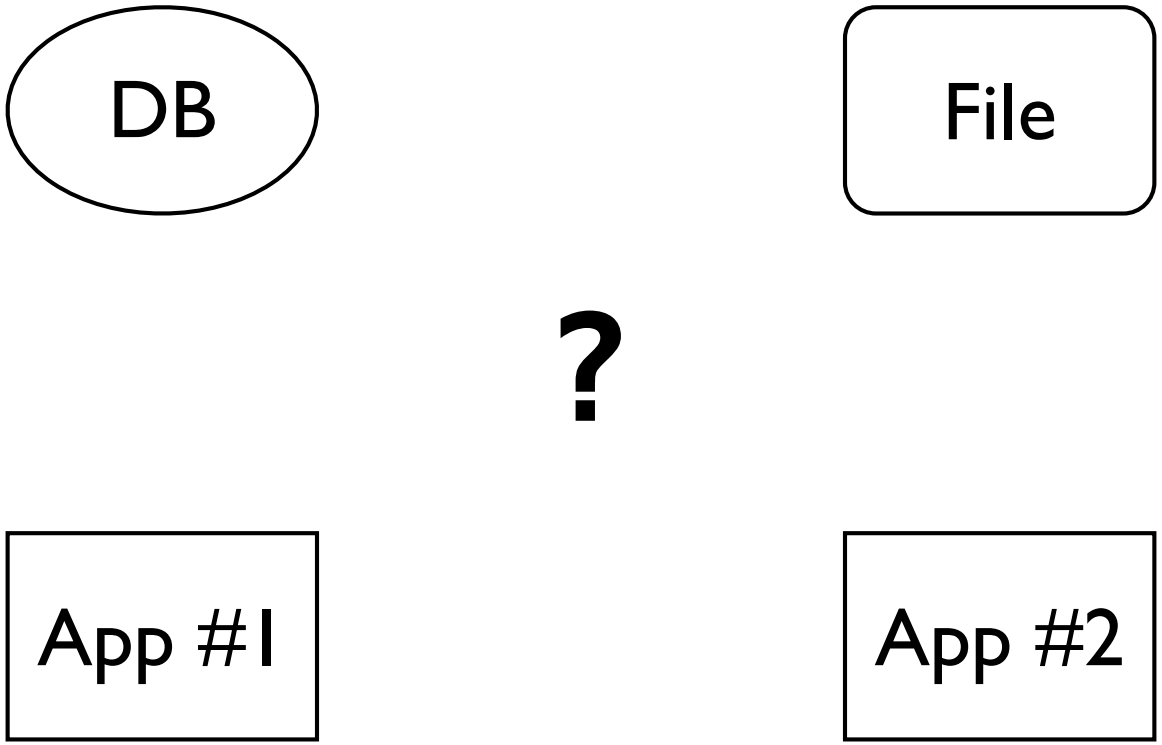
Suggested reading: Python module docs:

- os (environ, system, …)
- sys (argv, exit)
- getopt, optparse
- subprocess

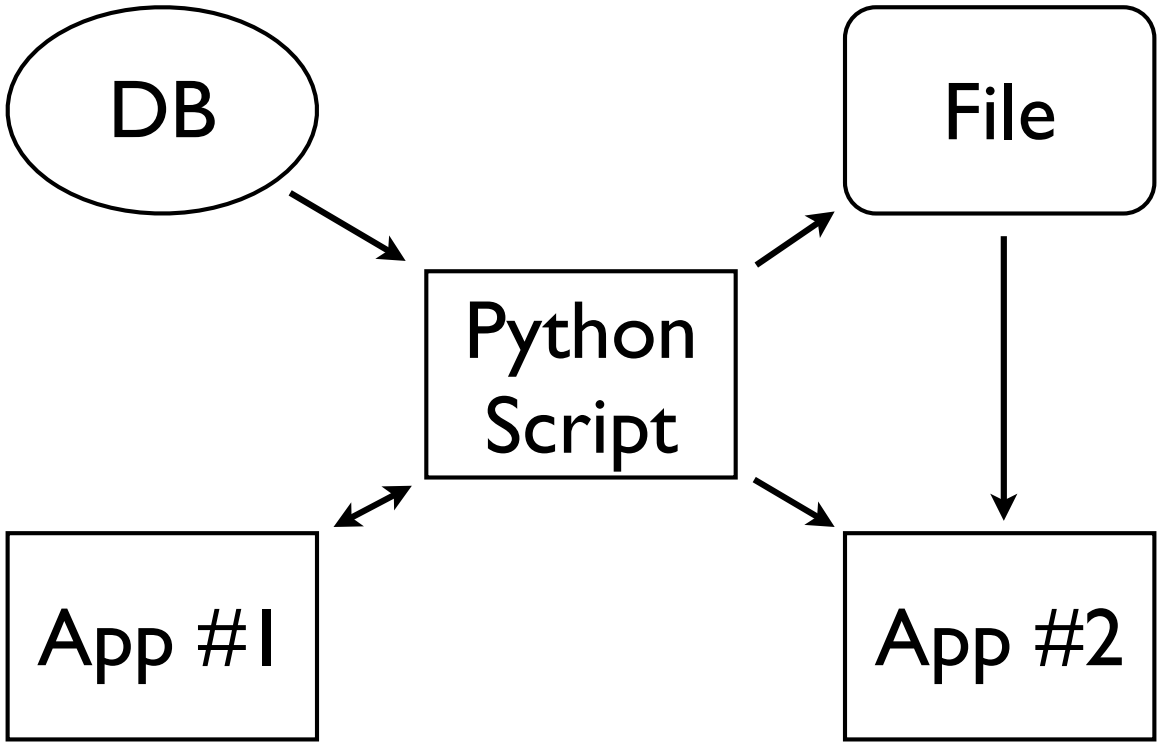
Turn In Homework

Homework Review

Problem



(One) Solution



Introducing … The Shell

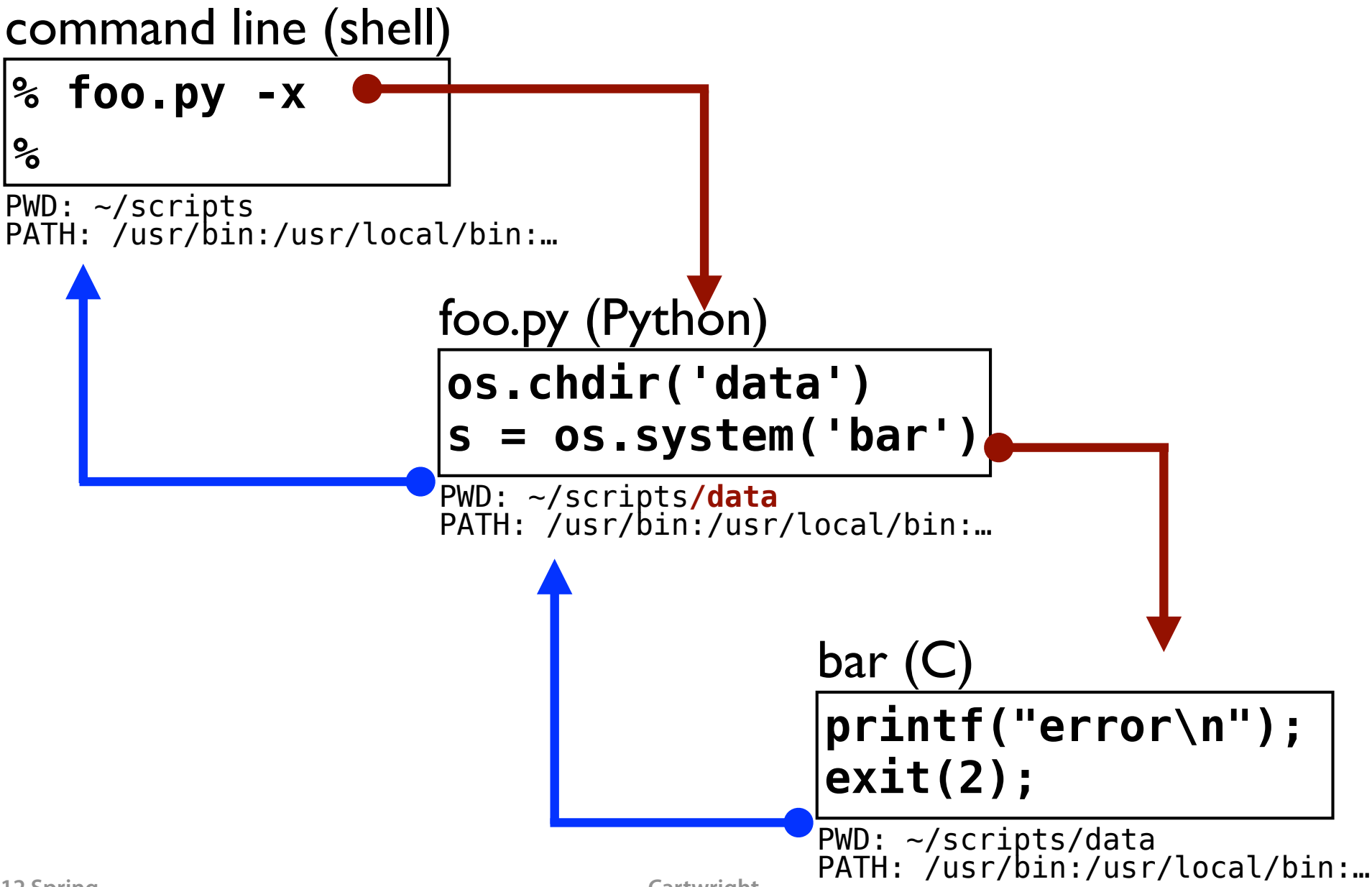
The Shell

- Is a ***program*** — e.g., **`/bin/sh`**
- Is an interpreted scripting language (like Python)
- Runs "commands", but most are separate programs
- Has variables, control structures, functions, …
- Gives control over input and output
- Supports basic workflows via I/O pipelines (|)
- Interactive (the command line) or scripted

Common Shell Commands

- Built-in: **`pwd`**, **`cd`**, **`echo`**
- Manual pages: **`man`**
- List files: **`ls`**, **`find`**
- Change file privileges: **`chown`**, **`chmod`**
- File manipulation: **`cp`**, **`mv`**, **`rm`**, **`ln`**, **`unlink`**
- Directory manipulation: **`mkdir`**, **`rmdir`**
- View files: **`cat`**, **`more`**, **`less`**, **`wc`**
- Filter and/or manipulate files: **`grep`**, **`sed`**, **`awk`**
- Text editors: **`pico`**, **`vi`**, **`emacs`**
- Scripting: **`python`**, **`perl`**, **`ruby`**

Shells and Commands



The System ⇒ Your Script

Command-Line Arguments

`sys.argv[0]`	command (script) name
`sys.argv[1:]`	command-line arguments

```
% python homework-07.py input-07-log.txt
```

```
if len(sys.argv) != 2:
    print 'Usage: %s FILE' % \
          (os.path.basename(sys.argv[0]))
    sys.exit(1)

log_file_name = sys.argv[1]

# Homework 7 code....
```

Better Argument Parsing

```
% python foo.py -ab --lo --with foo file1
```

```
import getopt

try:
    opts, args = getopt.getopt(
        sys.argv[1:], 'abc',
        ['lo', 'with='])

except getopt.GetoptError, e:
    usage()
    sys.exit(1)
```

```
opts: [('-a', ''), ..., ('--with', 'foo')]
args: ['file1']
```

Advanced Argument Parsing

```
% python noisy.py --quiet ...
```

```
from optparse import OptionParser

parser = OptionParser()
parser.set_defaults(be_quiet=False)
parser.add_option('-q', '--quiet',
                  dest='be_quiet',
                  action='store_true',
                  help='suppress output')
# ...
(options, args) = parser.parse_args()

if not options.be_quiet:
    print 'Welcome to my script!'
```

Environment

- `os.environ`
  - All environment variables available to script
  - Dictionary-like object: keys and values are strings
- Readable and writable

```
def not_sbin(path):
    return '/sbin' not in path

paths = os.environ['PATH'].split(':')
new_paths = filter(not_sbin, paths)
os.environ['PATH'] = ':'.join(new_paths)

if 'LD_LIBRARY_PATH' in os.environ:
    del os.environ['LD_LIBRARY_PATH']
```

Your Script ⇒ The System

Running a Command

```
exit_status = os.system('echo $PATH')
```

- Runs the given command in a new (sub)shell
- Command inherits **`os.environ`**, **`PWD`**, etc.
- Python waits for the command to finish
- Returns exit status (× 256) from the command

```
os.system('gzip ' + output_filename)
os.system('Rscript foo.R 1200 42')
os.system('octave eval.m > my_oct.log')
```

Sneaky system() Subtleties

Children do not affect parent

```
print os.getcwd()       # => /home/cat/foo
os.system('cd bar')
print os.getcwd()       # => ???
```

Quoting

```
os.system("echo 'Don'\\''t say \"no\"!'")
```

```
[shell]    echo 'Don'\''t say "no"!'
```

```
[output]           Don't say "no"!
```

Return Values and Errors

- Unix: (exit status × 256) + terminal signal (usu. 0)
- For a shell command, exit status of **0** is good

```
exit_status = os.system(command)
if exit_status != 0:
    print '"%s" failed:' % (command)
    signal = exit_status % 128
    if signal:
        print 'Signal:', signal
    else:
        print 'Exited:', exit_status / 256
```

- ***Windows is different!***

Beyond `os.system()`

Problems With `os.system()`

- Runs a subshell
  - One more program to run
  - Have to deal with quoting issues
  - May not want subshell to interpret arguments (e.g., **>**)
  - Potential security issues
- No input (stdin) or output (stdout, stderr)
  - Waits for input, when command prompts for input
  - All output goes to same place **`print`** does in Python
  - Myriad complicated solutions:

    `os.popen*()`, **`popen2.*`**, **`commands.*`**

One Module to Rule Them All…

Standard input

```
subprocess.Popen(args, bufsize=0,
executable=None, stdin=None, stdout=None,
stderr=None, preexec_fn=None,
close_fds=False, shell=False, cwd=None,
env=None, universal_newlines=False,
startupinfo=None, creationflags=0)
```

Much, much more…

subprocess.call()

```
es = subprocess.call(command, shell=True)
```

- Like **os.system()**, except with sensible exit status

```
cmd = 'echo $PATH'
es = subprocess.call(cmd, shell=True)
```

```
/usr/kerberos/sbin:/usr/kerberos/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
```

```
es = subprocess.call([command, ...])
```

- No subshell; command and arguments in list

```
cmd = ['echo', '$PATH']
es = subprocess.call(cmd)
```

```
$PATH
```

subprocess.Popen() and Output

```
proc = subprocess.Popen(command,
                        stdout=subprocess.PIPE,
                        stderr=subprocess.PIPE)
(output, error) = proc.communicate()
exit_status = proc.returncode
```

- Command and arguments in list… ***or***
- Add **`shell=True`** and then command is one string
- Standard output and standard error captured
- Get exit status from **`returncode`** attribute

Wrapper Function

```
def my_system(command, shell=False):
    p = subprocess.Popen(command,
                         stdout=subprocess.PIPE,
                         stderr=subprocess.PIPE,
                         shell=shell)
    (stdout, stderr) = p.communicate()
    return (p.returncode, stdout, stderr)
```

```
cmd = ['octave', 'analyze.m', '1234']
(status, stdout, stderr) = my_system(cmd)
if (status != 0) or ('error' in stderr):
    print 'Uh oh....'
# Do something with stdout
```

And In The End…

Exit a Script

```
sys.exit(1)
```

- Quits script immediately with given exit status
- Defaults to **0**, which means success
- End of script implies **sys.exit()**

```
try:
    if os.path.exists(filename):
        cmd = ['Rscript', filename]
        (stat, out, err) = my_system(cmd)
except OSError, e:
    print 'Failed to run R:'
    print 'Message:', e
    sys.exit(1)
```

Last 2 Slides…

Other Scripting Languages

- In command-line scripts, expect
  - Command-line arguments
  - Environment
  - Standard in, out, and error
  - System calls
  - Exit with status
  - Windows will always be different…
- Embedded scripting (PHP, Lua, JavaScript, …)
  - Expect more restrictions

Homework

- Run a simulation program lots of times (1000?)
- Collect output, reformat, and save to file
- Beginning programmers: Read the hints!
- Part 3 is optional… but results in a pretty graph