

Python: 7 Patterns for scripting

Matt Harrison

matthewharrison@gmail.com

<http://panela.blog-city.com/>

SCALE 2009

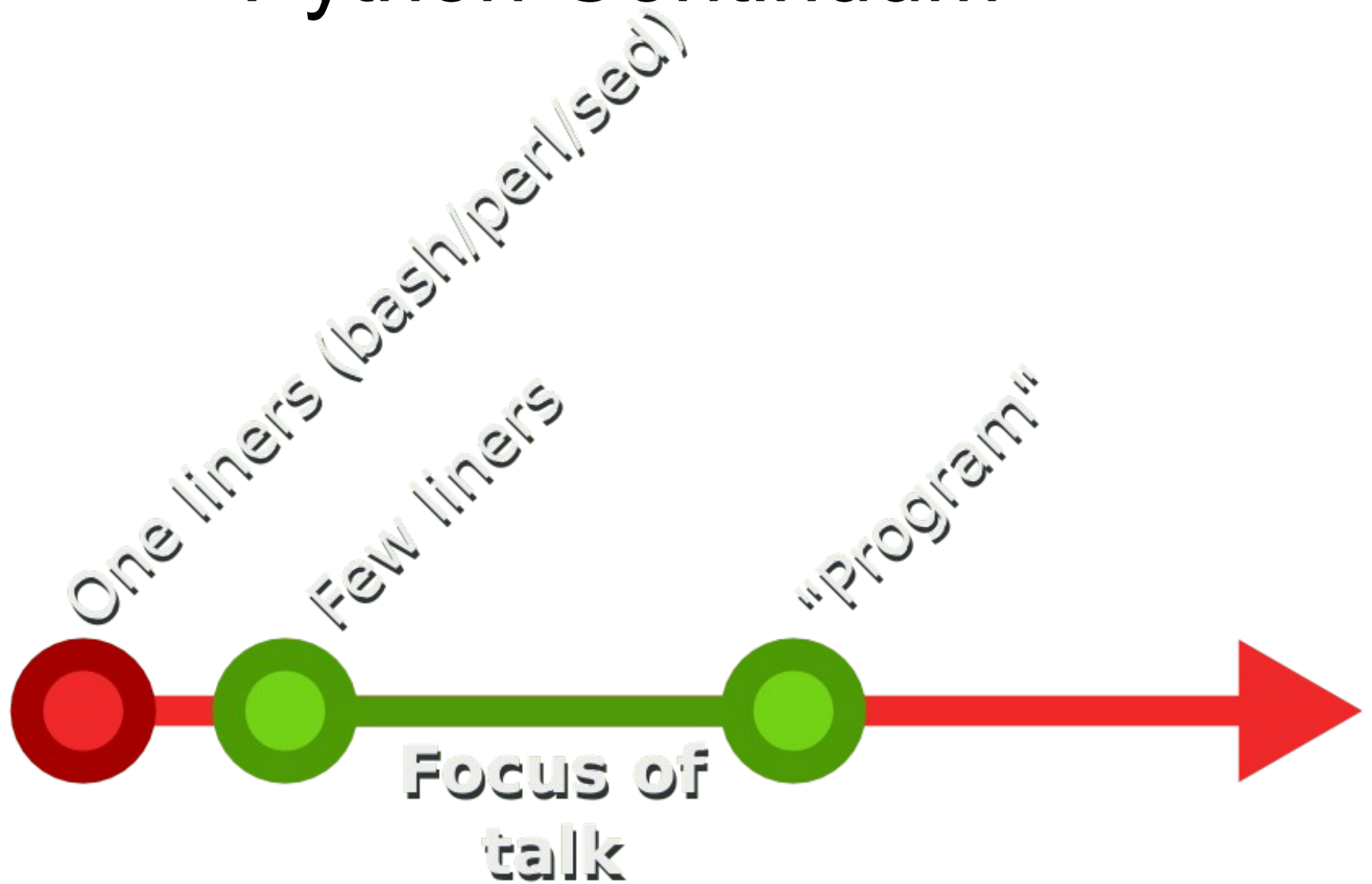
(C)2008, licensed under a [CC \(BY-SA\)](#) license.

Beginner level

Cliff's Notes

See handout and
poachplate.

Python Continuum



Focus on `stdlib` and
pure python

Why Python?

Advantages of Python

- Clean code
- 3rd party libs
- Cross platform
- Threading
- Datastructures (lists, dicts)
- Performance
- ...

Cons of Python

- Not as 'high level' as bash
- gluing scripts together is more tedious

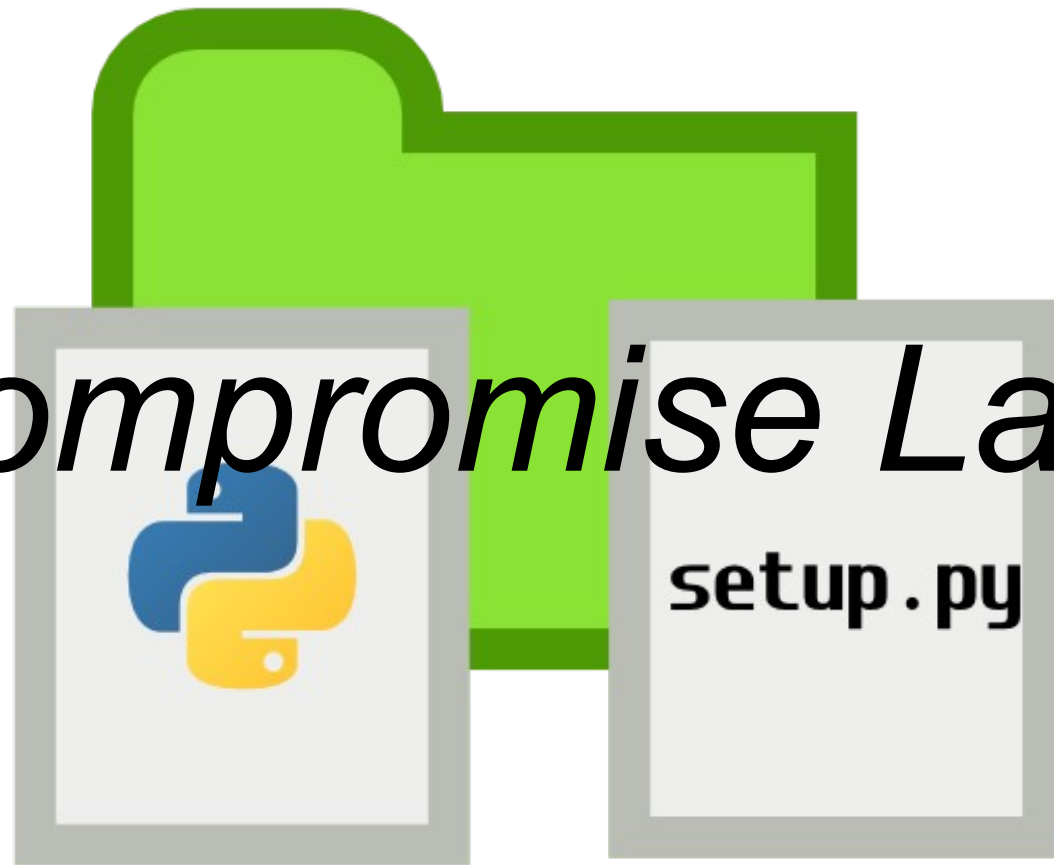

```
>>> import this
```

```
The Zen of Python, by Tim Peters
```

```
There should be one-- and preferably  
only one --obvious way to do it.
```

Scripting patterns in python

1 *Compromise Layout*



Scrapage/Packriipt

• `.py` (module/package)
vs script

PATH vs PYTHONPATH

PATH

- executable
- not (usually) importable

PATH vs PYTHONPATH

PYTHONPATH

- not (usually) executable
- importable

Compromise layout

Layout:

Project/

bin/

script(.py) (thin wrapper)

scriptlib/

__init__.py

scriptimpl.py

setup.py

2 Conditional main

```
#!/usr/bin/env python
```

```
copyright/license
```

```
Module docstring
```

```
Imports
```

```
Globals
```

```
Functions/classes
```

```
Executable main
```

```
if __name__ == '__main__':
```



Bad

```
>>> lines = open(sys.argv[1]).readlines()
>>> for line in lines:
...     ..
>>> # More stuff
>>> # Other stuff
>>> # No functions
>>> sys.exit()
```

Good practices

- Don't want behavior to change during import
- Globals are 'bad'
- Limit side effects
- move logic into grokable chunks (ie functions)

Conditional main

```
>>> # imports
>>> # classes/functions
>>> def main(prog_args):
...     # process args
...     # execute functions
...     # return exit code
>>> if __name__ == '__main__':
...     sys.exit(main(sys.argv))
```

sys.exit

Limit use

- *0* - Success
- *Non-Zero* - Error

Results

- Modular code
- Testable chunks
- Code can be imported/reused
- Easier to modify

Summarize

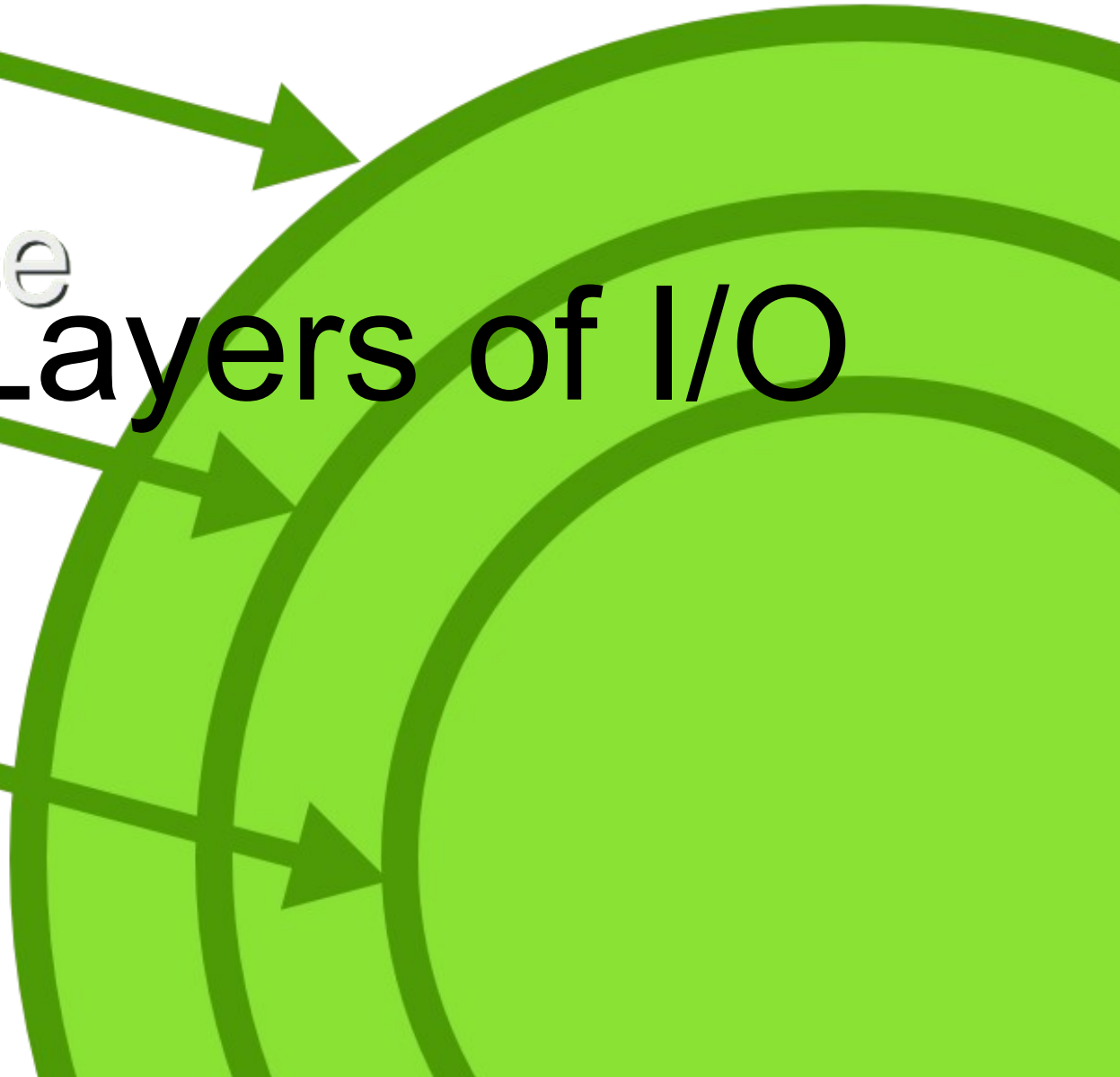
ok to have bad code for run once or if no one else is using it

main (filename)

file instance

generator

3 - 3 Layers of I/O



What interface?

- main - filename
- file-like
- generator

main

- accepts filenames (defaults to stdin/stdout)
- Do file exception handling here
- Do close of files

file-like

Can take `open()`, `sys.stdin`, `StringIO...`

- Testing is easier

Generator

- Efficient
(Also use when dealing with dbs)

Generators 101

```
>>> def num_list(count):  
...     results = []  
...     i = 0  
...     while i < count:  
...         results.append(i)  
...         i += 1  
...     return results
```

Generator:

```
>>> def num_gen(count):  
...     i = 0  
...     while i < count:  
...         yield i  
...         i += 1
```

Generators 101 (2)

```
>>> for num in num_list(3):
```

```
...     print num
```

Generator:

```
>>> for num in num_gen(3):
```

```
...     print num
```

Generators 101 (2)

```
>>> num_list(3)
[0, 1, 2]
>>> gen = num_gen(3)
>>> gen
<generator object at 0x7f78e133dcf8>
>>> gen.next()
0
>>> gen.next()
1
```

3 layers

```
>>> def gen_cat(line_iter):  
...     for line in line_iter:  
...         # business logic  
...         yield line  
>>> def file_cat(fin, fout):  
...     for line in gen_cat(fin):  
...         fout.write(line)
```


3 layers (cont)

```
>>> def main(pargs):  
...     # optparse blah blah...  
...     fin = sys.stdin  
...     if opt.fin:  
...         fin = open(opt.fin)  
...     fout = sys.stdout  
...     file_cat(fin, fout)
```

Testing generator

```
>>> list(gen_cat(['foo\n', 'bar\n']))  
['foo\n', 'bar\n']
```

Testing file

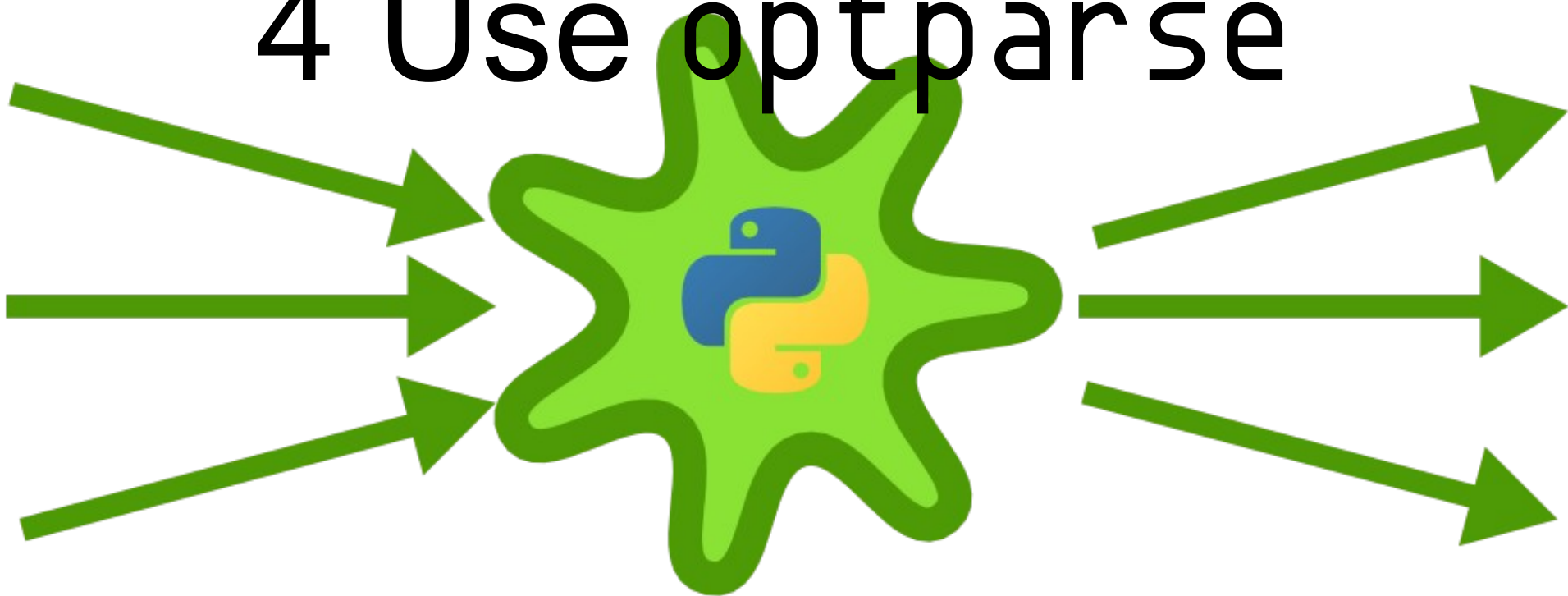
```
>>> import StringIO
>>> fout = StringIO.StringIO()
>>> file_cat(StringIO.StringIO('foo\nbar\nn'), fout)
>>> fout.getvalue()
foo
bar
```

Testing filename

```
>>> main(['--fin', '/tmp/foo', '--fout',  
         '/tmp/out'])  
>>> open('/tmp/out/').read()  
foo  
bar
```



4 Use optparse



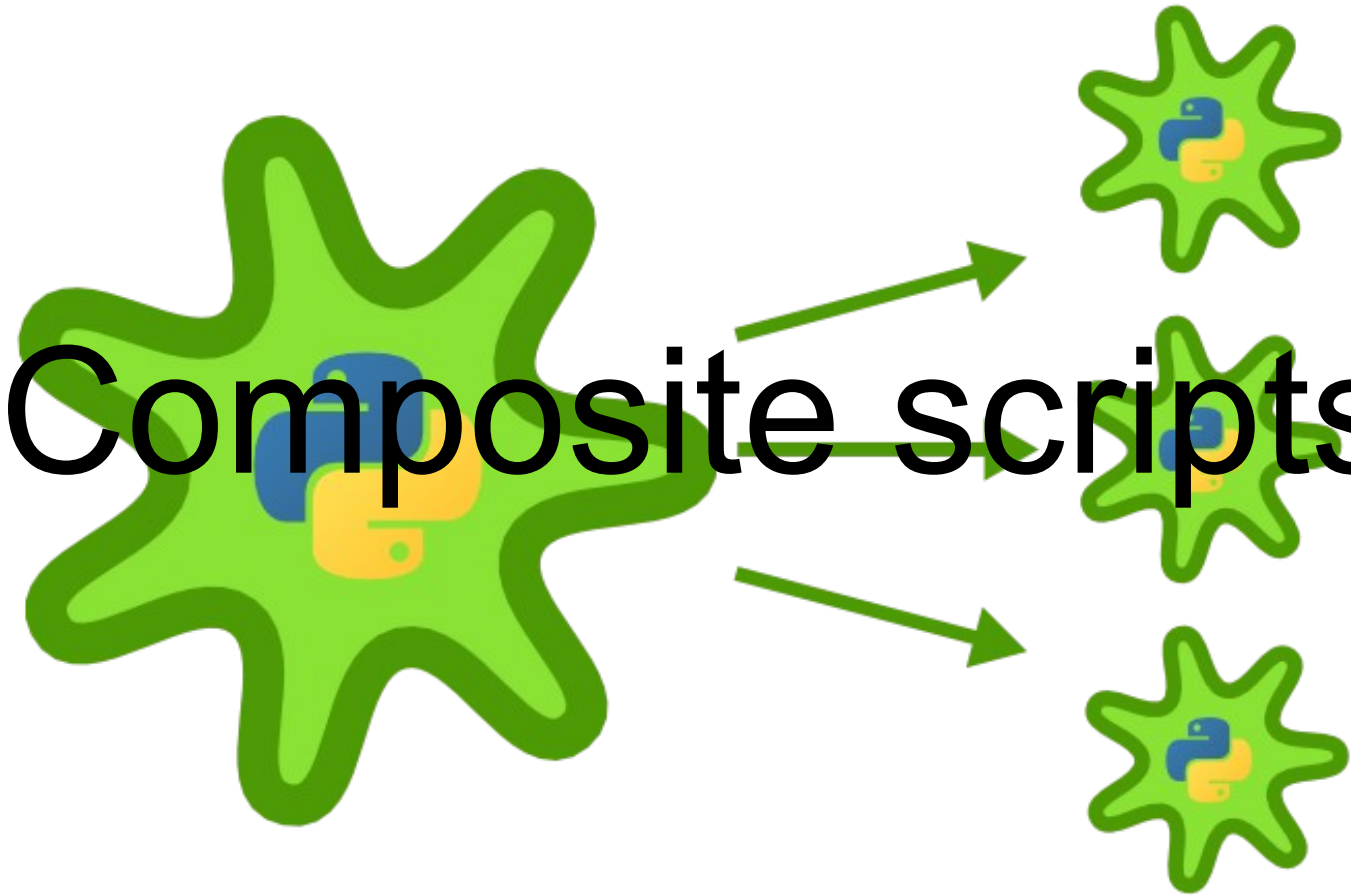
Commandline parsing options

- manual
- getopt
- optparse

optparse benefits

- Nice usage (*--help*)
- Provides *--version*

5 Composite scripts



Composite scripts

SVN style "script command --options"

```
>>> def main(pargs): # pargs =  
    ['script.py', 'status', '--some-option']  
...     if pargs[1] == 'status':  
...         status.main(pargs[2:])
```

Bonus

If you want to have scripts support this, you get it for free from complying with *Executable main* and *Use optparse*



6 Leave no trace

No print

How will I debug?

Use logging

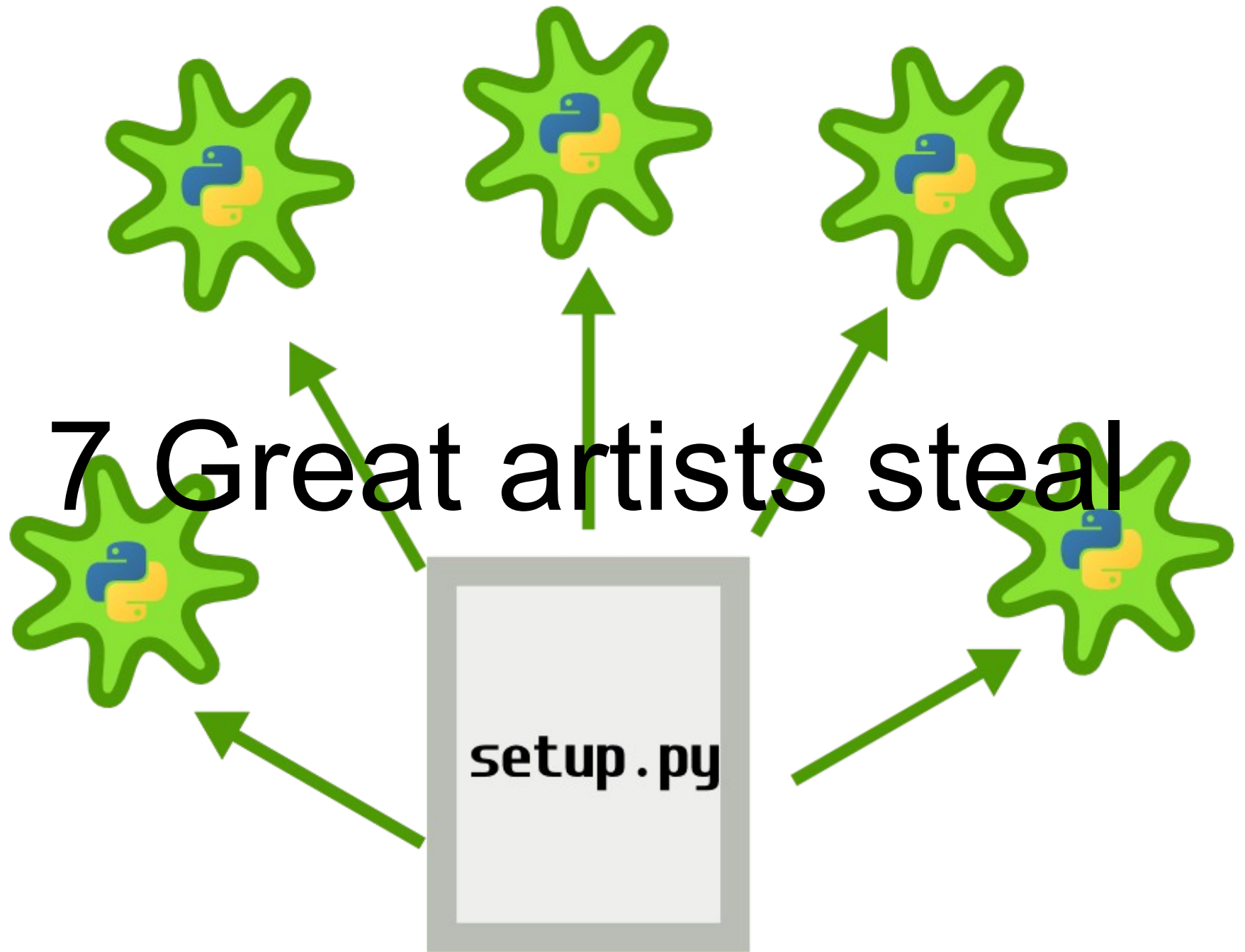
logging boilerplate

```
>>> LOGFILE =  
os.path.expanduser( '~/.script.log' )  
>>> logger =  
logging.getLogger( 'ScriptLogger' )  
>>> logger.setLevel( logging.DEBUG )  
>>> handler =  
handlers.RotatingFileHandler( LOGFILE,  
maxBytes=500, backupCount=2 )  
>>> log_format = Formatter( "%(asctime)s -  
%(name)s - %(levelname)s - %(message)s" )  
>>> handler.setFormatter( log_format )  
>>> logger.addHandler( handler )
```

atexit is also your
cleaning friend

Benefits

- Using 3 layers
- You'll have (proper) logging



setup.py

Bonus pattern: Bad

```
>>> from sys import *
```

Better

```
>>> import sys as s
```

Non-pattern: Testing

Side note

Code reviews are usually more effective than testing

Figure out how to test

- None
- Manually
- Automated
 - unittest style
 - doctest
 - input/output checking

Testing is easier with
well structured code

Globals make testing
hard

No testing makes
refactoring hard

No testing/refactoring ->

- crappy code
- harder to add features

crappy code -> unhappy
co-workers

poachplate

poachplate

- Compromise Layout
- Executable main
- Theft Packaging

handout

- Verbose file organization
- support for Unix configuration hierarchy
- tempfile
- Script chaining
- pid file
- logging

Thanks:

- docutils
- OOo
- inkscape
- pygments

Handout at <http://panela.blog-city.com/>