
PyCon 2016
CHINA

Python Hooking, Patching and Injection

丁来强 (Lai Qiang Ding)

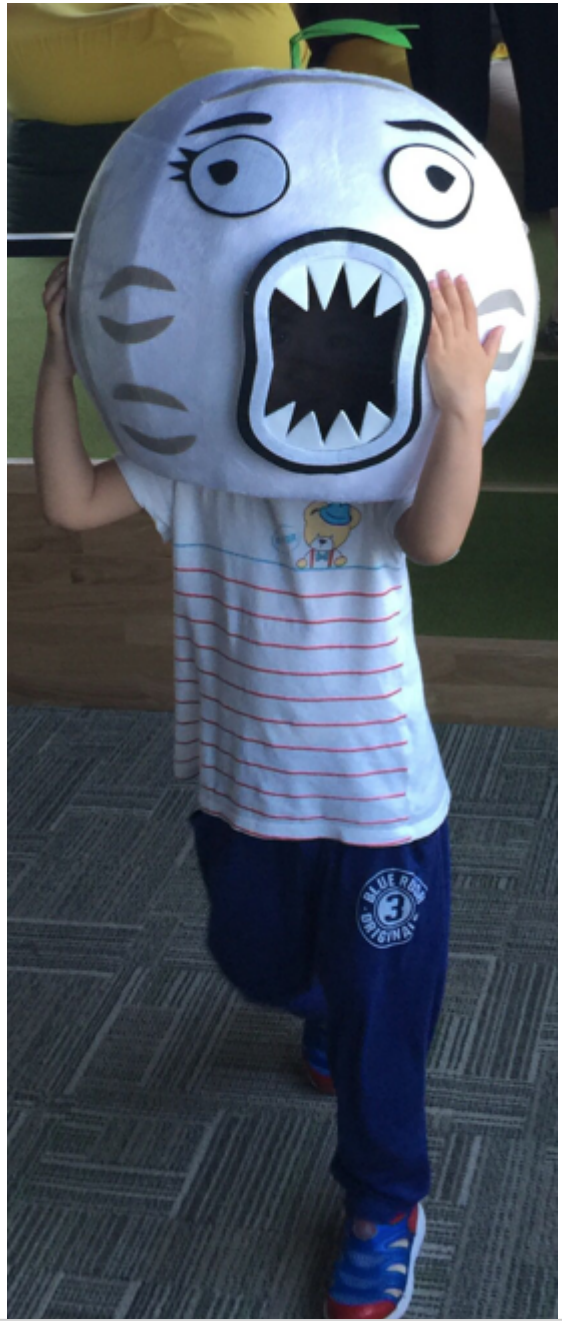
wjo1212

wjo1212@163.com

LaiQiangDing

About Me

Father of a 4 years' boy



About Me

- Worked for 10+ years.
- @Splunk

Agenda

- Hook Technology, Use cases and Patterns
  - Monkey patch
  - System level hook
  - Language level hook

- Interpreter level hook

Hook Technology

- Object and Mutability
  - Monkey patch
- System level hook
  - Import hook, sys trace
- Language level hook
  - Context manager, Magic methods, Descriptor
  - Decorator, Meta class
- Interpreter level hook
  - AST, Bytecodes, Frame Inspect

Use Cases

- DRY = Don't repeat yourself
- Monkey Patching
- Unit Test
- Supportability
  - Logging
  - Troubleshooting, Debugging
  - Error Handling
  - Auto-fixing

Use Cases (con't)

- Dynamic Analysis/Scaning
  - Violation check
  - Best practice/Standard Check
  - Coverage
  - Performance Profiling
  - APM
- Performance Improvement

- Lazy evaluation, lazy import
- Some other cool things

You will learn

- Cool Hook Technology and mechanisms, and major helpful use cases and patterns
- Knows the pros and cons to use the hook technology and know in what kind of scenarios use them
- Know how to further learn Python Hooking technology systematically

1. Object and Mutability

Almost everything in Python are objects

Example: how to intercept std IO (print to buffer instead of console)?

In [29]:
```
# make print to a buffer rather than console?

print "abc"
print "xyz"
print "123"
```

```
abc
xyz
123
```

In [23]:
```
import sys
from StringIO import StringIO
output = StringIO()
```

In [24]:
```
# patching
stdout = sys.stdout
sys.stdout = output
```

In [25]:
```
print "abc"
print "xyz"
print "123"
```

no results... actually written to buffer

In [26]:
```
output.seek(0)
output.readlines()
```

Out[26]:
```
['abc\n', 'xyz\n', '123\n']
```

recover

In [27]:
```
sys.stdout = stdout
```

In [28]:
```
print "abc"
print "xyz"
print "123"
```

```
abc
xyz
123
```

more

In [ ]:
```
sys.stderr
sys.stdin
sys.displayhook
```

Example: How to audit all file opening (when, who and how open)?

In [ ]:
```
open("./t1.txt", "w").close()
open("./t2.txt", "w").close()
open("./t3.txt", "w").close()
open("./t4.txt", "w").close()
```

In [31]:
```
import time
open_file_history = []

real_open = __builtins__.open

def my_open(name, *args, **kwargs):
    open_file_history.append(time.strftime("%H:%M:%S") + '" open "' + name + '"')  # *
    print("detected file opening: " + name)

    return real_open(name, *args, **kwargs)

__builtin__.open = my_open
```

In [32]:
```
open("./t1.txt", "w").close()
open("./t2.txt", "w").close()
open("./t3.txt", "w").close()
open("./t4.txt", "w").close()
```

```
detected file opening: ./t1.txt
detected file opening: ./t2.txt
detected file opening: ./t3.txt
detected file opening: ./t4.txt
```

In [31]:
```
!ls t*
```

```
t1.txt t2.txt t3.txt t4.txt
```

In [17]:
```
# now get the open file history for further purpose
open_file_history
```

Out[17]:
```
['19:49:24" open "./t1.txt"',
 '19:49:24" open "./t2.txt"',
 '19:49:24" open "./t3.txt"',
 '19:49:24" open "./t4.txt"']
```

recover

In [33]:
```
__builtins__.open = real_open
```

Example: Function is object and mutable

In [23]:
```
def fn(): pass

dir(fn)[24:]
```

Out[23]:
```
['func_closure',
 'func_code',
 'func_defaults',
 'func_dict',
 'func_doc',
 'func_globals',
 'func_name']
```

In [24]:
```
fn.hello = 100
fn.word = "abc"

dir(fn)[24:]
```

Out[24]:
```
['func_closure',
 'func_code',
 'func_defaults',
 'func_dict',
 'func_doc',
 'func_globals',
 'func_name',
 'hello',
 'word']
```

In [25]:
```
del fn.hello

dir(fn)[24:]
```

Out[25]:
```
['func_closure',
 'func_code',
 'func_defaults',
 'func_dict',
 'func_doc',
 'func_globals',
 'func_name',
 'word']
```

2. System level hook

2.1 Import Hook

import path

```
def import_module(mod_name):
    if mod_name in sys.modules:
        return sys.modules[mod_name]

    for dir_name in sys.path:
        print dir_name
        file_name = op.join(dir_name, mod_name + ".

        m = _exec_file(mod_name, file_name)
        sys.modules[mod_name] = m
        return m

    raise ImportError("...")
```

In [31]:
```
import sys
sys.modules.keys()[:10]
```

Out[31]:
```
['email.MIMEAudio',
 'IPython.core.error',
 'ipython_genutils.py3compat',
 'traitlets.config.sys',
 'ipykernel.parentpoller',
 'traitlets.config.decorator',
 'ctypes.os',
 'pexpect.select',
 'runpy',
 'gc']
```

2.1.1 how to make a external module importable externally (w/o code change)?

In [34]:
```
ret = !ls import_hook_lib
ret
```

Out[34]:
```
['hello_pycon.py',
 'hello_pycon.pyc',
 'hello_world.py',
 'hello_world.pyc',
 'urllib2.py',
 'urllib2.pyc']
```

In [30]:
```
import hello_world
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-30-bb2a1ee2b7fa> in <module>()
----> 1 import hello_world

ImportError: No module named hello_world
```

2.1.1.1 by $PYTHONPATH

Icon:

means: need to restart kernel. shortcut: typing zero twice: 0 0

In [35]:
```
!cat import_hook_lib/hello_world.py
```

```
print "hello world!"
```

In [36]:
```
!echo 'PYTHONPATH: '$PYTHONPATH
```

```
PYTHONPATH:
```

In [37]:
```
!python -c "exec(\"import sys\\nimport hello_world\")"
```

```
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "<string>", line 2, in <module>
ImportError: No module named hello_world
```

In [38]:
```
!export PYTHONPATH="./import_hook_lib"; python -c "exec(\"import sys\\nimport hello_world\")"
```

```
hello world!
```

2.1.1.1 via site.py

In [40]:
```
ret = !ls import_hook_lib
ret
```

Out[40]:
```
['hello_pycon.py',
 'hello_pycon.pyc',
 'hello_world.py',
 'hello_world.pyc',
 'urllib2.py',
 'urllib2.pyc']
```

In [41]:
```
!rm -f /Users/wjo1212/pycon2016/lib/python2.7/site-packages/pycon.pth
!echo "/Users/wjo1212/Documents/Private/MySharing/PyCon2016/PythonHacking/Demo/import_hook_lib\\nimport hello_wor
!cat /Users/wjo1212/pycon2016/lib/python2.7/site-packages/pycon.pth
```

```
/Users/wjo1212/Documents/Private/MySharing/PyCon2016/PythonHacking/Demo/import_hook_lib
import hello_world
```

In [42]:
```
# note the   default "hello world"
!python -c "pass"
```

```
hello world!
```

In [43]:
```
!python -c "import hello_pycon"
```

```
hello world!
hello pycon !
```

clean

In [1]:
```
!rm -f /Users/wjo1212/pycon2016/lib/python2.7/site-packages/pycon.pth
!cat /Users/wjo1212/pycon2016/lib/python2.7/site-packages/pycon.pth
```

```
cat: /Users/wjo1212/pycon2016/lib/python2.7/site-packages/pycon.pth: No such file or directory
```

2.1.2 how many ways to import a library?

In [46]:
```
!cat ./load_me.py
```

```
print("load me: I'm loaded!")
```

method 1: use import keyword

directly put module name into local namespaces

In [47]:
```
import load_me
```

```
load me: I'm loaded!
```

method 2: use __import__ function

In [48]:
```
load_me = __import__('load_me1')
```

```
load me1: I'm loaded!
```

method 3: use imp

could by pass sys.modules, import

(deprecated in Py3 by import_lib)

In [34]:
```
import sys, load_me
'load_me' in sys.modules
```

Out[34]:
```
True
```

In [50]:
```
import imp

name = 'load_me'
fp, pathname, description = imp.find_module(name)

try:
    json = imp.load_module(name, fp, pathname, description)
finally:
    if fp:
        fp.close()

# note: the module is loaded again
```

```
load me: I'm loaded!
```

method 4: use import_lib

replace imp in Py3 (recommended in Py3 than __import__)

In [52]:
```
import importlib
json = importlib.import_module('load_me')
```

2.1.3 How to audit module importing?

(when, who and how imported)

In [ ]:
```
# how to monitor those importing?

# file1.py
import md5
import json
import socket

# file2.py
import requests
import functools
```

Hook __import__

prepare

In [1]:
```
import importlib, imp

def imp_load(name):
    fp, pathname, description = imp.find_module(name)

    try:
        json = imp.load_module(name, fp, pathname, description)
    finally:
        if fp:
            fp.close()
```

hook __import__

In [2]:
```
impl = __import__

def my_importer(*args, **kwargs):
    print("** importing: " + args[0])
    return impl(*args, **kwargs)

__builtins__.__import__ = my_importer
```

import, import_lib are hooked,

imp is NOT hooked

In [3]:
```
import os

importlib.import_module('sys')

imp_load('load_me')   # not hooked
```

```
** importing: os
** importing: sys
load me: I'm loaded!
```

```
Note: import and import_lib use __impor__  internally
```

clean

In [4]:
```
__builtins__.__import__ = impl
```

2.1.4 how to replace installed module with local version?

In [5]:
```
!ls import_hook_lib/url*
print "-" * 30
!cat ./import_hook_lib/urllib2.py
```

```
import_hook_lib/urllib2.py  import_hook_lib/urllib2.pyc
------------------------------

def urlopen(*args, **kwargs):
    print("dummy urlopen: {}".format(str(args)))
    print("** do something cool **")
    return args[0]
```

add sys path

In [3]:
```
import sys
sys.path.insert(0, './import_hook_lib')   # Note: position 0
```

urllib2 is hooked

In [4]:
```
import urllib2
urllib2.urlopen('http://localhost:8888/notebooks')
```

```
dummy urlopen: ('http://localhost:8888/notebooks',)
** do something cool **
```

Out[4]:
```
'http://localhost:8888/notebooks'
```

2.1.5 How to import a non-existing module by providing temporary ones?

In [3]:
```
class PyCon(object):
    def __str__(self):
        return "hello PyCon China 2016!"

import pycon2016  # if not exist, provide a object of PyCon()
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-3-b285bcf3c6a0> in <module>()
      3         return "hello PyCon China 2016!"
      4
----> 5 import pycon2016  # if not exist, provide a object of PyCon()

ImportError: No module named pycon2016
```

hook sys.meta_path

Yet, another hooking method as __import__

In [1]:
```
import sys
class Watcher(object):
    @classmethod
    def find_module(cls, name, path, target=None):
        print("Imorting: {}".format(name))
        return None  # Note: bypass to other Finder/Loader

# insert into sys.meta_path
sys.meta_path.insert(0, Watcher)
```

In [2]:
```
import urllib2
```

```
Imorting: urllib2
Imorting: httplib
Imorting: mimetools
Imorting: rfc822
```

In [2]:
```
import sys

# note the PyCon class
class PyCon(object):
    def __str__(self):
        return "hello PyCon China 2016!"

class MyModuleLoader(object):
    @classmethod
    def find_module(cls, name, path, target=None):
        if name == 'pycon2016':
            return cls  # Note
        return None

    @classmethod
    def load_module(cls, name):
        if name == 'pycon2016':
            return PyCon()
        raise ImportError("pycon")

# insert into sys.meta_path
sys.meta_path.insert(0, MyModuleLoader)
```

In [6]:
```
import pycon2016
print pycon2016
```

```
hello PyCon China 2016!
```

More:

how to auto-install a library that doesn't exist?

2.1.6. How to support a remote virtual repo in sys.path?

In [ ]:
```
import sys
sys.path.append("box://mycompany.repo.com/repo/team")

import some_module   #  automatically import the some_module from box repo
```

hook sys.path_hook

for specific file path, folder or repo (e.g. zip)

In [1]:
```
import sys
sys.path_hooks
```

Out[1]:
```
[zipimport.zipimporter]
```

In [2]:
```
[p for p in sys.path if 'zip' in p]   # note the zip file
```

Out[2]:
```
['/Users/wjo1212/pycon2016/lib/python27.zip']
```

Examples: one path hook to support lib hosted on remote repo (e.g. AWS s3)

In [ ]:
```
import sys
class Dummy(object):
    def __str__(self):
        return "Hello PyCon 2016"

class S3ImportParser(object):
    KEY = 's3://'
    def __init__(self, path_entry):
        if path_entry.startswith(self.KEY):
            print "Handle: " + path_entry
            self.path_repo = path_entry
            return

        # raise ImportError means passing to other Loader
        raise ImportError()

    def find_module(self, fullname, path=None):
        print 'Search for "%s" on %s' % (fullname, self.path_repo)
        return self

    def load_module(self, fullname):
        print 'Load for "%s"' % fullname
        return Dummy()
```

In [1]:
```
sys.path_hooks.append(S3ImportParser)

# suppose there's a S3 repo in sys.path
sys.path.append("s3://mycompany.repo.com/repo/team")

import some_module

print "-" * 30
print str(some_module)
```

```
Handle: s3://mycompany.repo.com/repo/team
Search for "some_module" on s3://mycompany.repo.com/repo/team
Load for "some_module"
------------------------------
Hello PyCon 2016
```

2.2 System Hook

2.2.1 How to hook system exist event and do some clean-up?

exit hook - atexit

In [ ]:
```
# %load sys_hook/exit_hook.py
from threading import Thread, current_thread

import atexit

def exit0(*args, **kwarg):
    print '** exit0', current_thread().getName(), args, kwarg

def exit1():
    print '** exit1', current_thread().getName()
    raise Exception, 'exit1'

def exit2():
    print '** exit2' , current_thread().getName()
```

In [ ]:
```
atexit.register(exit0, 1, 2, a=1)
atexit.register(exit1)
atexit.register(exit2)

@atexit.register
def exit3():
    print '** exit3', current_thread().getName()

if __name__ == '__main__':
    print '** main', current_thread().getName()
```

In [6]:
```
!python sys_hook/exit_hook.py
```

```
** main MainThread
** exit3 Dummy-1
** exit2 Dummy-1
** exit1 Dummy-1
Error in atexit._run_exitfuncs:
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/atexit.py", line 24, in _run_exitfuncs
    func(*targs, **kargs)
  File "sys_hook/exit_hook.py", line 10, in exit1
    raise Exception, 'exit1'
Exception: exit1
** exit0 Dummy-1 (1, 2) {'a': 1}
Error in sys.exitfunc:
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/atexit.py", line 24, in _run_exitfuncs
    func(*targs, **kargs)
  File "sys_hook/exit_hook.py", line 10, in exit1
    raise Exception, 'exit1'
Exception: exit1
```

Note:

1. Sequence: LIFO
2. Threading Context: dummy-1
3. Exception: overcome
4. parameters

2.2.2. How to know if some specific exceptions happens?

- when, who and how
- Note: even the exception is captured

sys.trace

capture all system events

Examples: capture all exception events

In [ ]:
```
# %load md_exception.py
try:
    dct  = {}
    a = dct['abc']
except KeyError:
    pass

print "***complete import: ", __file__
```

In [1]:
```
import sys, traceback

stop = False
def trace(frame, event, args):
    if stop:
        return

    if event == "exception":
        print "capture exception: ", args[0], args[1]
        print "----" * 10

    return trace

sys.settrace(trace)

import md_exception

stop = True
```

```
capture exception:  <type 'exceptions.KeyError'> ('abc',)
----------------------------------------
***complete import:  md_exception.pyc
```

Examples: hack calling stack via settrace

In [ ]:
```
# %load hack_fn.py
import sys

def trace_func(frame,event,arg):
    if "a" not in frame.f_locals:
        return

    value = frame.f_locals["a"]
    if value % 2 == 0:
        value += 1
        frame.f_locals["a"] = value

def print_odd(a):
    print a

if __name__ == "__main__":
    sys.settrace(trace_func)
    for i in range(0, 6):
        print_odd(i)
```

In [4]:
```
!python hack_fn.py
```

```
1
1
3
3
5
5
```

Note:

Coverage, Profiler and implemented basing on sys.settrace

3. Language level hook (syntax sugar)

3.1 How to control a block's enter and exit

- logging, timer, resource control etc

Context manager

Provide hook a "Enter" especially "Exit" event for a block of code

Even the exception happens in the block

In [2]:
```
import time
class MyTimer(object):
    def __init__(self, tag='default'):
        self.tag = tag

    def __enter__(self,):
        self.start = time.time()

    def __exit__(self, exc, val, trace):
        self.end = time.time()
        print '*** performance for "' + self.tag + '" ***'
        print "    {} seconds".format(self.end - self.start)
        return True
```

In [3]:
```
def test1(x, max):
    try:
        1/(max-x-1)
    except ZeroDivisionError as e:
        raise

def test2(x, max):
    1/(max-x-1)


with MyTimer("function call"):
    for x in range(10000000):
        test1(x, 10000000)

print "-" * 30

with MyTimer("function call with try block"):
    try:
        for x in range(10000000):
            test2(x, 10000000)
    except ZeroDivisionError as ex:
        raise
```

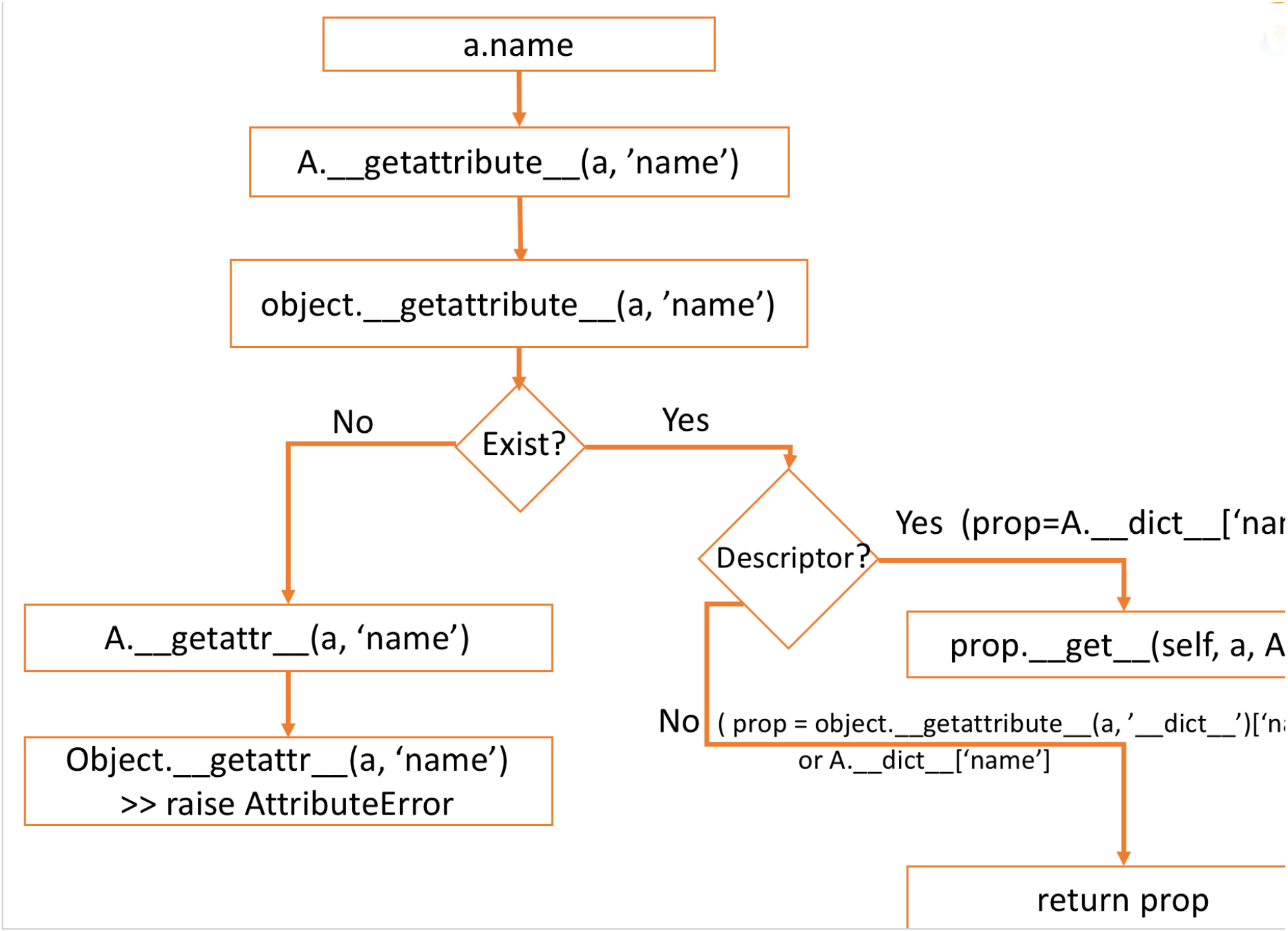
```
*** performance for "function call" ***
    3.11267113686 seconds
------------------------------
*** performance for "function call with try block" ***
    2.97663712502 seconds
```

Note:

1. __exit__ is called even exception happens (return True to overcome)
2. quite suitable for resource management and `scope management`

3.2 Magic Methods

Provide hook all kinds of operations around an object


a.name
A.__getattribute__(a, 'name')
object.__getattribute__(a, 'name')
Exist?
No
Yes
A.__getattr__(a, 'name')
Object.__getattr__(a, 'name')
>> raise AttributeError
Descriptor?
Yes (prop=A.__dict__['nar
prop.__get__(self, a, A
No
(prop = object.__getattribute__(a, '__dict__')['n
or A.__dict__['name']
return prop

3.2.1 How to support "Lazy property" via Chain calling

Attributes Access

In [ ]:
```
# Note: consider a 10GB table
students = AzureTable('user1.storage.azure.com/table/students')

# Only get the data when be accessed
print students.xiaoMing.Address.data
print students.Jim.birthday.data
```

In [13]:
```
class AzureTable(object):
    def __init__(self, url, row=None, col=None, level='table'):
        self.__url = url
        self.__row = row
        self.__col = col
        self.__level = level

    def _fetch(self):
        assert self.__level == 'col'
        print '*** Downloading from\n\t"{}/{}/{}"'.format(self.__url,
                                                         self.__row,
                                                         self.__col)
        print '*** Downloading Complete'
        return "Hello PyCon 2016: {}.{}".format(self.__row, self.__col)

    def __getattr__(self, item):
        if self.__level == 'table':
            return AzureTable(self.__url, row=item, level='row')
        elif self.__level == 'row':
            return AzureTable(self.__url, row=self.__row, col=item, level='col')

        if item == 'data':
            return self._fetch()
```

In [14]:
```
students = AzureTable('user1.storage.azure.com/table/students')

print students.xiaoMing.Address.data
print "-" * 30
print students.Jim.birthday.data
```

```
*** Downloading from
        "user1.storage.azure.com/table/students/xiaoMing/Address"
*** Downloading Complete
Hello PyCon 2016: xiaoMing.Address
------------------------------
*** Downloading from
        "user1.storage.azure.com/table/students/Jim/birthday"
*** Downloading Complete
Hello PyCon 2016: Jim.birthday
```

3.2.2 How to auditting property access for an 3rd party made object?

and how to provide default value for 3rd party object with limited __slot__?

In [ ]:
```
from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])

p = Point(x=1, y=2)

print p.x    # How to monitor the access?
print p.y

# print p.z    # How to provide default value?
# p.z = 10     # Note: cannot do this!
```

In [13]:
```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])

def monitor(self, item):
    print "** accessing :" + item
    return super(Point, self).__getattribute__(item)

Point.__getattribute__ = monitor

p = Point(x=1, y=2)
print p.x
print p.y

def handle_non_exist(self, item):
    print "** handling :" + item
    return str(item)

Point.__getattr__ = handle_non_exist

print p.z
```

```
** accessing :x
1
** accessing :y
2
** accessing :z
** handling :z
z
```

3.2.3 How to validate property value setting for an object transparently?

Descriptors

In [ ]:
```
p = People(age=1, height=200)   # allow
p.age = 10   # allow!


p = People(age=-1, height=200)     # disallow!
p.age = 0    # disallow!
```

Descriptor is a more low-granularity access control

In [14]:
```
class IntegerProperty(object):
    def __init__(self, mi=None, mx=None):
        self.min = mi
        self.max = mx

    def __get__(self, obj, objtype):
        return self.val

    def __set__(self, obj, val):
        if (self.min is None or self.min <= val) \
            and (self.max is None or val <= self.max):
            self.val = val
        else:
            raise ValueError("value is out of range")

    # def __delete__(...)
```

In [16]:
```
class People(object):
    age = IntegerProperty(1, 130)
    height = IntegerProperty(1, 230)
    def __init__(self, age, height):
        self.age = age
        self.height = height

p = People(age=1, height=200)

p.age = 10

# Demo: do some typing here

#p.age = 0
#p = People(age=-1, height=200)
#del p.age
```

3.2.3 Operator overwritting

Example: How to support Scala Lambda in Python?

In [ ]:
```
map(_ * 2, xrange(4))

# get: [0, 2, 4, 6]

map(10 + _, xrange(4))

# get: [10, 11, 12, 13]
```

In [10]:
```
class Call(object):
    def __mul__(self, other):
       return lambda a: a * other

    def __radd__(self, other):
        return lambda a: a + other

_ = Call()

print map(_ * 2, xrange(4))

print map(10 + _, xrange(4))
```

```
[0, 2, 4, 6]
[10, 11, 12, 13]
```

Example: How to support stream stype workflow operation?

In [ ]:
```
t1, t2, t3, t4, t5, t6, t7 = (Task('t' + str(x)) for x in range(1,8))

t1 >> t2 >> t3 >> t4
t2 >> t5 >> t6
t1 >> t7

Dag.draw_dag(t1)

# get:
"""
   + t1
     + t2
       + t3
         + t4
       + t5
         + t6
     + t7
"""
```

In [ ]:
```
class Task(object):
    def __init__(self, name):
        self.name = name
        self.post_tasks = []

    def __rshift__(self, task):
        self.post_tasks.append(task)
        return task

    def __str__(self):
        return self.name
```

In [11]:
```
t1, t2, t3, t4, t5, t6, t7 = (Task('t' + str(x)) for x in range(1,8))

t1 >> t2 >> t3 >> t4
t2 >> t5 >> t6
t1 >> t7

class Dag(object):
    @staticmethod
    def draw_task(task, level):
        print "  " * (level + 1) + " + " + str(task)

    @staticmethod
    def draw_dag(task, level=0):
        Dag.draw_task(task, level)
        for n in task.post_tasks:
            Dag.draw_dag(n, level+1)

Dag.draw_dag(t1)
```

```
   + t1
     + t2
       + t3
         + t4
       + t5
         + t6
     + t7
```

3.3.1 How to monitor function calling and do something like logging, timing?

In [ ]:
```
import wrapt, time

def logger(prefix='*'):
    @wrapt.decorator
    def _logger(fn, instance, args, kwargs):
        print '{} Enter "{}"'.format(prefix, fn.func_name)
        ret = fn(*args, **kwargs)
        print '{} Exit "{}"'.format(prefix, fn.func_name)
        return ret

    return _logger

@wrapt.decorator
def timer(fn, instance, args, kwargs):
    t1 = time.time()
    ret = fn(*args, **kwargs)
    print "* Time consumed: {} seconds".format(time.time() - t1)
    return ret
```

In [13]:
```
@timer
@logger("+")
def do_job():
    time.sleep(0.5)
    print "Hello PyCon 2016 China!"

do_job()
```

```
+ Enter "do_job"
Hello PyCon 2016 China!
+ Exit "do_job"
* Time consumed: 0.50387597084 seconds
```

3.3.2. How to make the function more robust by overcome some exceptions?

In [14]:
```
import wrapt

@wrapt.decorator
def ignore_error(fn, instance, args, kwargs):
    try:
        return fn(*args, **kwargs)
    except Exception as e:
        print "# Skip errors: {}".format(e)

@ignore_error
def do_job():
    print "Hello PyCon 2016 China!"
    raise  ValueError("invalid")

do_job()
```

```
Hello PyCon 2016 China!
# Skip errors: invalid
```

3.3.3. how to add features to some functions like check auth for page rendering?

In [ ]:
```
class HomePage(object):
    need_login = False

    @check_auth
    def show(self):
        print "** This is Home Page."

class AdminPage(object):
    need_login = True  # check session key
    session_key = ''

    @check_auth
    def show(self):
        print "** This is Admin Page."
```

In [ ]:
```
import wrapt

@wrapt.decorator
def check_auth(fn, instance, args, kwargs):
    if getattr(instance, 'need_login', False) \
            and not getattr(instance, 'session_key', ''):
        print "**** Permission Denied: {}".format(type(instance))
        return # Do nothing

    return fn(*args, **kwargs)
```

In [16]:
```
p1 = HomePage()
p2 = AdminPage()

p1.show()
p2.show()
```

```
** This is Home Page.
**** Permission Denied: <class '__main__.AdminPage'>
```

3.3.4. how to change function behavior: e.g. directly make it dummy?

In [17]:
```
import wrapt

@wrapt.decorator
def dummy(fn, instance, args, kwargs):
    return "Do nothing"

@dummy
def do_job():
    time.sleep(0.5)
    print "Hello PyCon 2016 China!"

do_job()
```

Out[17]:
```
'Do nothing'
```

3.4 Class Decorator

Provide hook for class construction/method calling

3.4.1. How to simply make a class's methods thread-safe?

In [1]:
```
import time
class Task(object):
    def __init__(self):
        self.data = "xxx"

    def run1(self):
        data = self.data
        time.sleep(0.5)
        self.data = data + "111"

    def run2(self):
        data = self.data
        time.sleep(0.5)
        self.data = data + "222"

    def run3(self):
        data = self.data
        time.sleep(0.5)
        self.data = data + "333"
```

In [2]:
```
from threading import Thread

t = Task()

ts = [Thread(target=lambda j: j.run1(), args=(t, )),
         Thread(target=lambda j: j.run2(), args=(t, )),
         Thread(target=lambda j: j.run3(), args=(t, ))]

[s.start() for s in ts]
[s.join() for s in ts]

print "final data:", t.data
assert len(t.data) == 12, "t.data length should be 12 but is " + str(len(t.data))
```

```
final data: xxx222

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-2-426ac35e53c4> in <module>()
     11
     12 print "final data:", t.data
---> 13 assert len(t.data) == 12, "t.data length should be 12 but is " + str(len(t.data))

AssertionError: t.data length should be 12 but is 6
```

In [18]:
```
import wrapt, time, threading, inspect
from threading import Thread

def synchronized(cls):
    lock = threading.RLock()

    @wrapt.decorator
    def _wrapper(fn, instance, args, kwargs):
        with lock:
            return fn(*args, **kwargs)

    for k, v in cls.__dict__.iteritems():
        if not k.startswith("__") and inspect.isfunction(v):
            setattr(cls, k, _wrapper(v))

    return cls
```

In [ ]:
```
@synchronized
class Task(object):
    def __init__(self):
        self.data = "xxx"

    def run1(self):
        data = self.data
        time.sleep(0.5)
        self.data = data + "111"

    def run2(self):
        data = self.data
        time.sleep(0.5)
        self.data = data + "222"

    def run3(self):
        data = self.data
        time.sleep(0.5)
        self.data = data + "333"
```

In [26]:
```
t = Task()

ts = [Thread(target=lambda j: j.run1(), args=(t, )),
      Thread(target=lambda j: j.run2(), args=(t, )),
      Thread(target=lambda j: j.run3(), args=(t, ))]

[s.start() for s in ts]
[s.join() for s in ts]

print "final data:", t.data
assert len(t.data) == 12, "t.data length should be 12 but is " + str(len(t.data))
```

```
final data: xxx111222333
```

about Decorator Overhead

use optimized wrapt for relative low performance impact

3.6 Metaclass

Provide hook for class construction (somehow metaprogramming)

3.6.1 How to monitor class generation (in class hierarchy)

when, how, who and easily get all sub-class for a base class

In [6]:
```
class RegisterLeafClasses(type):
    def __init__(cls, name, bases, nmspc):
        super(RegisterLeafClasses, cls).__init__(name, bases, nmspc)
        if not hasattr(cls, 'registry'):
            cls.registry = set()
        cls.registry.add(cls)
        cls.registry -= set(bases) # Remove base classes

    def __str__(cls):
        if cls in cls.registry:
            return cls.__name__
        return cls.__name__ + ": " + ", ".join([sc.__name__ for sc in cls.registry])

class Color(object):
    __metaclass__ = RegisterLeafClasses
```

In [7]:
```
class Blue(Color): pass
class Red(Color): pass
class Green(Color): pass
class Yellow(Color): pass
print(Color)
class PhthaloBlue(Blue): pass
class CeruleanBlue(Blue): pass

print(Color)
```

```
Color: Blue, Yellow, Red, Green
Color: Red, CeruleanBlue, PhthaloBlue, Yellow, Green
```

3.6.2 How to make a class as final (cannot be inherited)

In [8]:
```
class final(type):
    def __init__(cls, name, bases, namespace):
        super(final, cls).__init__(name, bases, namespace)
        for klass in bases:
            if isinstance(klass, final):
                print "**debug** ", name, bases, namespace
                raise TypeError(str(klass.__name__) + " is final")

class A(object):
    pass

class B(A):
    __metaclass__= final
```

In [9]:
```
# compile error cause B is final
class C(B):
    pass
```

```
**debug**  C (<class '__main__.B'>,) {'__module__': '__main__'}

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-cc4042472a21> in <module>()
      1 # compile error cause B is final
----> 2 class C(B):
      3     pass

<ipython-input-8-191c1a0aea9e> in __init__(cls, name, bases, namespace)
      5             if isinstance(klass, final):
      6                 print "**debug** ", name, bases, namespace
----> 7                 raise TypeError(str(klass.__name__) + " is final")
      8
      9 class A(object):

TypeError: B is final
```

3.6.3 Another way to decorate class: Example: Syncrhonized metaclass

In [24]:
```
import threading, inspect, collections

import wrapt

class SynchronizedClass(type):
    @classmethod
    def __prepare__(name, bases, **kwds):
        return collections.OrderedDict()

    def __new__(metacls, name, bases, namespace, **kwds):
        ret = type.__new__(metacls, name, bases, dict(namespace))

        ret.lock = threading.RLock()
        @wrapt.decorator
        def _wrapper(fn, instance, args, kwargs):
            with ret.lock:
                return fn(*args, **kwargs)

        for k, v in namespace.iteritems():
            if not k.startswith("__") and inspect.isfunction(v):
                setattr(ret, k, _wrapper(v))

        return ret
```

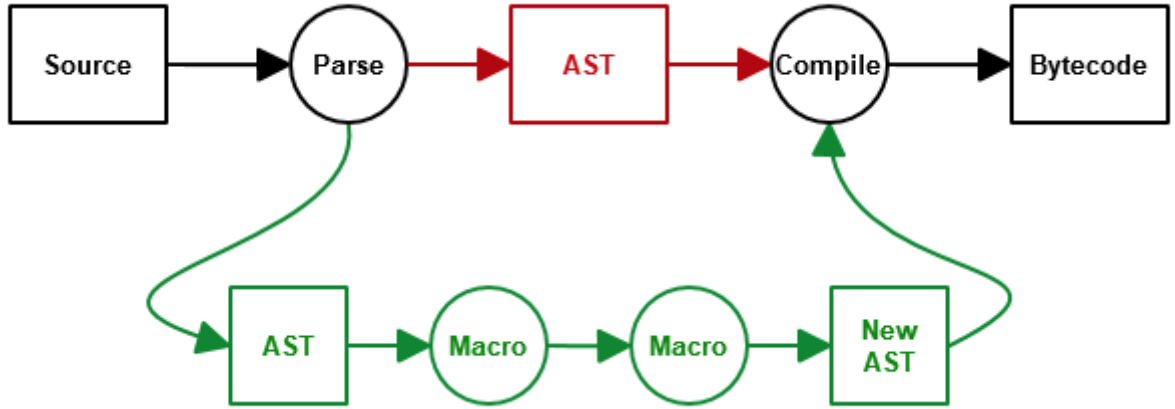
In [25]:
```
class Task(object):
    __metaclass__ = SynchronizedClass   # Note

    def __init__(self):
        self.data = "xxx"

    def run1(self):
        data = self.data
        time.sleep(0.5)
        self.data = data + "111"

    def run2(self):
        data = self.data
        time.sleep(0.5)
        self.data = data + "222"

    def run3(self):
        data = self.data
        time.sleep(0.5)
        self.data = data + "333"

t = Task()

ts = [Thread(target=lambda j: j.run1(), args=(t, )),
         Thread(target=lambda j: j.run2(), args=(t, )),
         Thread(target=lambda j: j.run3(), args=(t, ))]
[s.start() for s in ts]
[s.join() for s in ts]

print "final data:", t.data
assert len(t.data) == 12, "t.data length should be 12 but is " + str(len(t.data))
```

```
final data: xxx111222333
```

4. Interpreter Level Hook

4.1. AST

Provide hook method to change the Syntax tree



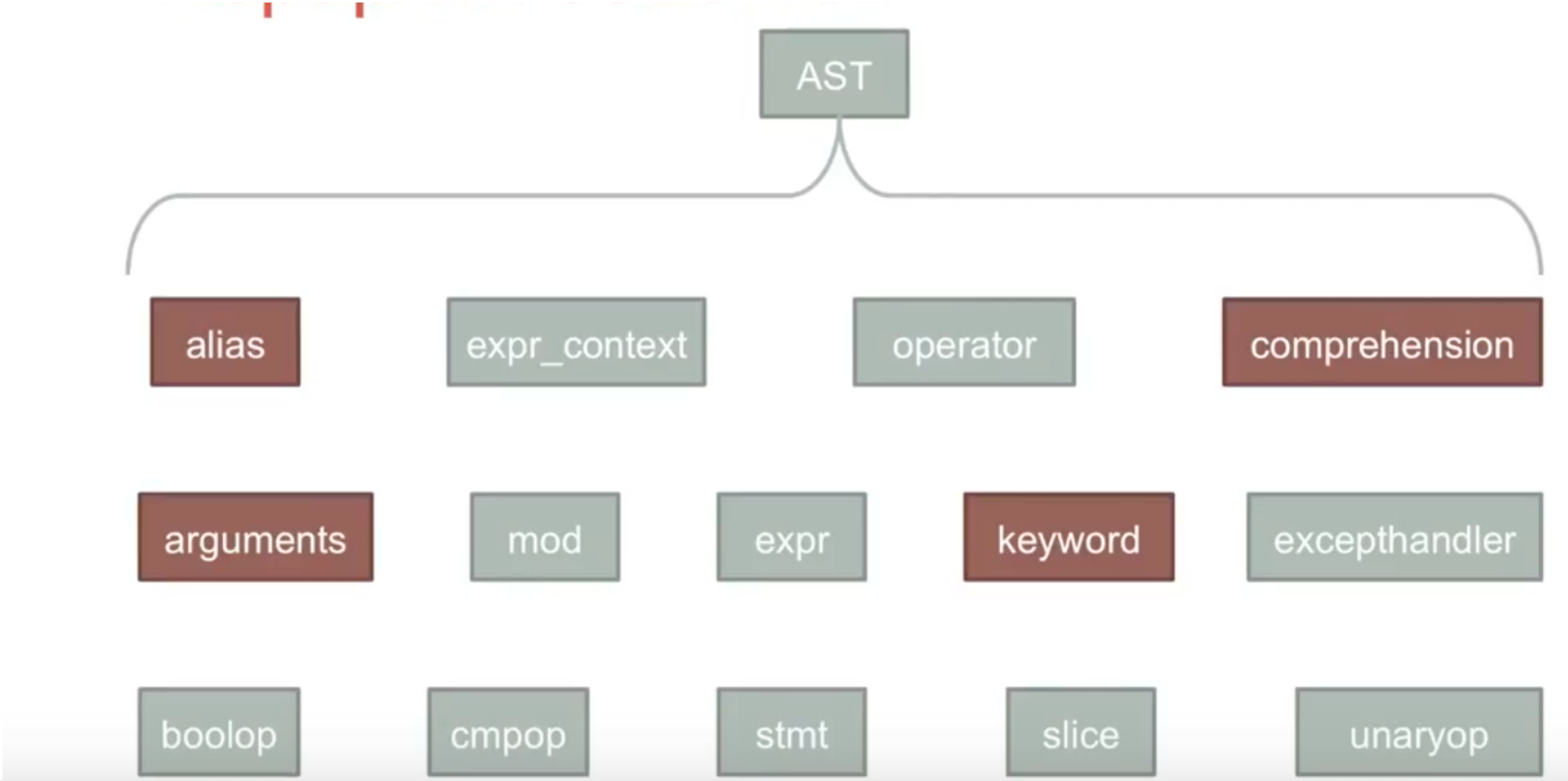
Basic

In [13]:
```
import inspect, ast, astunparse

def add(s1, s2):
    return s1 + s2
```

In [14]:
```
# get a pretty-printed dump of the AST
print(astunparse.dump(ast.parse(inspect.getsource(add))))
```

```
Module(body=[FunctionDef(
  name='add',
  args=arguments(
    args=[
      Name(
        id='s1',
        ctx=Param()),
      Name(
        id='s2',
        ctx=Param())],
    vararg=None,
    kwarg=None,
    defaults=[]),
  body=[Return(value=BinOp(
    left=Name(
      id='s1',
      ctx=Load()),
    op=Add(),
    right=Name(
      id='s2',
      ctx=Load()))),],
  decorator_list=[])])
```

Bytecodes elements



4.1.1 How to support Haskell pattern match syntax?

In [30]:
```
def factorial():
    if 0: 1
    if n is int: n * factorial(n-1)
    if []: []
    if [x] + xs: [factorial(x)] + factorial(xs)
    if {'n': n, 'f': f}: f(factorial(n))

assert factorial(0) == 1
assert factorial(5) == 120
assert factorial([3,4,2]) == [6, 24, 2]
assert factorial({'n': [5, 1], 'f': sum}) == 121
```

In [30]:
```
from patterns import patterns, Mismatch

@patterns
def factorial():
    if 0: 1
    if n is int: n * factorial(n-1)
    if []: []
    if [x] + xs: [factorial(x)] + factorial(xs)
    if {'n': n, 'f': f}: f(factorial(n))

assert factorial(0) == 1
assert factorial(5) == 120
assert factorial([3,4,2]) == [6, 24, 2]
assert factorial({'n': [5, 1], 'f': sum}) == 121
```

4.1.2. how to write an smart assert?

In [ ]:
```
smart_assert 3**2 + 4**2 != 5**2

"""
Assert Failed
3**2 -> 9
4**2 -> 16
3**2 + 4**2 -> 25
5**2 -> 25
3**2 + 4**2 != 5**2 -> False
"""
```

In [8]:
```
!cat ./macropy_case/smart_asserts.py
```

```
from macropy.tracing import macros, require
try:
    require[3**2 + 4**2 != 5**2]
except AssertionError as e:
    print e


try:
    a = 10
    b = 2
    with require:
        a > 5
        a * b == 20
        a < 2
except AssertionError as e:
    print e
```

In [ ]:
```
# import macropy.console
```

In [4]:
```
import macropy_case.smart_asserts
```

```
Require Failed
3**2 -> 9
4**2 -> 16
3**2 + 4**2 -> 25
5**2 -> 25
3**2 + 4**2 != 5**2 -> False
Require Failed
a < 2 -> False
```

4.2. Bytecodes

4.2.1 is it possible to hack the string "hello world"?

In [6]:
```
def hello():
    print "hello world"

hello()
```

```
hello world
```

In [9]:
```
import dis
from byteplay import Code, LOAD_CONST

c = Code.from_code(hello.__code__)
print c.code
```

```
  2           1 LOAD_CONST           'hello world'
              2 PRINT_ITEM
              3 PRINT_NEWLINE
              4 LOAD_CONST           None
              5 RETURN_VALUE
```

In [10]:
```
print c.code[0]
print c.code[1]
```

```
(SetLineno, 2)
(LOAD_CONST, 'hello world')
```

In [12]:
```
c.code[1] = (LOAD_CONST, "hello pycon 2016!")

hello.__code__ = c.to_code()

hello()
```

```
hello pycon 2016!
```

4.3. Frame Object

4.3.1 how to support interpolate string?

In [ ]:
```
name = 'Guido van Rossum'
places = 'Amsterdam', 'LA', 'New York', 'DC', 'Chicago',

s = """My name is ${'Mr. ' + name + ', Esquire'}.

I have visited the following cities:  ${', '.join(places)}.
"""

print s
```

In [ ]:
```
# output
"""
My name is Mr. Guido van Rossum, Esquire.

I have visited the following ci ties:  Amsterdam, LA, New York, DC, Chicago.
"""
```

In [12]:
```
import sys, re

def getchunks(s):
    matches = list(re.finditer(r"\$\{(.*?)\}", s))

    if matches:
        pos = 0
        for match in matches:
            yield s[pos : match.start()]
            yield [match.group(1)]
            pos = match.end()
        yield s[pos:]

def interpolate(templateStr):
    #framedict = sys._getframe(1).f_locals

    result = ''
    for chunk in getchunks(templateStr):
        if isinstance(chunk, list):
            result += str(eval(chunk[0]))
        else:
            result += chunk

    return result
```

In [13]:
```
name = 'Guido van Rossum'
places = 'Amsterdam', 'LA', 'New York', 'DC', 'Chicago',

s = """My name is ${'Mr. ' + name + ', Esquire'}.

I have visited the following cities:  ${', '.join(places)}.
"""

print interpolate(s)
```

```
My name is Mr. Guido van Rossum, Esquire.

I have visited the following cities:  Amsterdam, LA, New York, DC, Chicago.
```

More Use Cases

1. DRY (Don't repeat yourself)

2. Monkey Patching

- general usage
- especially when 3rd party module not changable

3. Unit Test (*)

In [10]:
```
# %load mut.py
import conf_loader as cl

def logic():
    config = cl.load()  # load settings from some REST

    # do a lot of complex settings

    return "abc"
```

In [11]:
```

    return "...."
```

In [12]:
```
import mut

def test_case_1():
    assert mut.logic() == "abc", "test failed"
    print "test passed"

test_case_1()
```

```
** conf_loader: load data from remote system....
test passed
```

what if the conf_loader.load:

- not available (complex system)
- hard to configure test data
- slow as heavy operation

In [ ]:
```
# %load conf_loader.py
def load():
    print("** conf_loader: load data from remote system....")

    # raise ValueError("time-out: remote system no response")

    # takes long time to get data
    return "...."
```

In [8]:
```
import mock
import mut

@mock.patch("conf_loader.load")
def main_v1(mock_load):
    mock_load.return_value = '{"settings":"..."}'

    assert mut.logic() == "abc", "test failed"
    print "test passed"

main_v1()
```

```
test passed
```

4. Supportability

4.1. Logging

4.2. Debugging, Troubleshooting

4.3. Error Handling by changing target behaviour

4.4. Auto-fixing (ditto)

5. Dynamic Analysis

5.1. Violation, Best practice or Standard Check

- How to detect if importing some risky module? like md5, SSL v2.0
- How to detect if do some risky operations? like removing files from system folder?

5.2. Coverage

5.3. Memory, Performance Profiling

5.4. APM

6. Performance Improvement

6.1. Lazy evaluation, lazy import

7. Some other cool things

Now you learned

- Cool Hook Technology and mechanisms, and major use cases to use them
- Know how to learn them further systematically
- Knows the pros and cons to use the hook technology and know in what kind of scenarios use them

wjo1212

wjo1212@163.com

LaiQiangDing