Working In Python

Brett Viren

Physics Department

BROOKHAVEN NATIONAL LABORATORY

2^{nd} Offline Tutorial 2008/12/14

Outline

Objective and Overview

Basic Python

Python and CMT and Gaudi and `nuwa.py`

Lesson Objective

After this lesson you should be able to:

1. Understand basic Python language for programing and interactive use.
2. Understand the difference between configure and run time code in a job.
3. Create CMT packages that use Python.
4. Accessing Gaudi interfaces from and writing Gaudi algorithms in Python.
5. What Python features `nuwa.py` provides.

Objective and Overview

Basic Python

Python and CMT and Gaudi and `nuwa.py`

Generalities

Python:

- Is an interpreted[1], dynamically typed, static (lexically) scoped language with a minimalistic syntax.
- Relies on **indentation delineated blocks** (you'll get over it!). Please, always use indentation of 4 spaces.
- Supports imperative, object oriented, and functional programming as well as reflection and meta-programming
- All data are objects including numerical and string literals.
- Comments start with # and go to end of line

Try it!:

```
# interactive python:
shell> ipython
In [1]: print 'Hello World!'
Hello World!
```

```
# batch python:
shell> cat hello.py
print "Hello World!"
shell> python hello.py
Hello World!
```

[1] Can also be compiled.

Basics: functions and branching

- No special meaning of single vs. double quotes. ''' or """ allow for multi line strings.
- Usual C++ assignment, equality, arithmetic operators and prefix **func(arg)** type function calls
- Print to stdout via **print** operator
- Define functions with **def func(arg,argwithdef=0):**
- Branch with **if/elif/else**, loop with **for x in list:**

```
def func(x,y=42):
    if x == y:
        print 'You got the answer'
    else: print 'Wrong, answer was',y
    return y
a = func(42)          # --> 42
You got the answer
a = func(42,43)       # --> 43
Wrong, answer was 43
```

More data types, lists and dictionaries

```
# lists:
lst = [1,2,'three']
if lst[0] == 1: print 'correct'
if lst[-1] == 'three': print 'correct'
for x in lst[0:2]: print x
1
2
# Dictionaries
dict = {'a':1, 'b':2, 'c':3}
dict['b']         # --> 1
for k,v in dict.iteritems():
    print k,v
a 1
b 2
c 3
dict.keys()     # --> ['a', 'c', 'b']
```

More data types, strings

Python strings are very powerful and easy to use.

```
s = 'Hello World!'
s.split(' ')                # --> ['Hello', 'World!']
'+'.join(['a','b','c'])     # --> 'a+b+c'
s[:-1] + ', Universe'       # --> 'Hello World, Universe'
s[0:s.find(' ')]            # --> 'Hello'
'  spaces  '.strip()        # --> 'spaces'
print '''"Multi-line strings are delimited by triple
single "'" or double '"' quotes and can contain other quotes"'''
"Multi-line strings are delimited by triple
single "'" or double '"' quotes and can contain other quotes"
```

Classes

- Define a class with **class ClassName:**
- Constructor is a method called **__init__(self)**
- The **self** is analogous to C++'s `this` pointer and must always be the first argument in a method definition[2]
- When **self** is used to call a method from a method it must be done like **self.method()** not **method(self)**

```
class Foo():
    def __init__(self,x):
        self.number = x
        self.bar()
    def bar(self):
        print 'number is',self.number
        return
f = Foo(42) # --> instance of Foo
number is 42
f.bar()     # --> None
number is 42
```

[2]Unless it is a "static" or class scoped method.

Modules

- Group code by directory and file.
- Modules are objects and can have methods and data members.
- Modules brought in to code via **import**

```
# MyMod.py
def func(): print 'Hello World!'
class Foo(): pass
foo = Foo()

# somecode.py
import MyMod
MyMod.func()
Hello World
f = MyMod.Foo()      # Instance of class Foo
from MyMod import Foo, foo
f = Foo()            # Instance of class Foo
foo                  # Default instance of class Foo
```

Any problems running the examples?

Objective and Overview

Basic Python

Python and CMT and Gaudi and nuwa.py

Adding Python to a CMT package

Directories:

- Modules go in `Package/python/Package/`
- No-install scripts go in `Package/share/`
- To-install scripts go in `Package/scripts/`

`cmt/requirements` file:

```
# To install modules:
apply_pattern install_python_modules

# To install scripts:
apply_pattern install_scripts
```

Packaging to Access C++ classes from Python

cmt/requirements file:

```
apply_pattern reflex_dictionary dictionary=PackageName \
              headerfiles=../dict/headers.h \
              selectionfile=../dict/classes.xml
```

dict/headers.h file:

```
#include "PackageName/ClassA.h"
#include "PackageName/ClassB.h"
#include "PackageName/ClassC.h"
```

dict/classes.xml file:

```
<lcgdict>
  <class name="ClassA"/>
  <class name="ClassB"/>
  <class name="ClassC"/>
  <!-- If need STL containers of these classes: -->
  <class name="std::vector&lt;ClassA*&gt;"/>
</lcgdict>
```

Simple Python Algorithm

```
from GaudiPython.GaudiAlgs import GaudiAlgo
from GaudiPython import SUCCESS, FAILURE

class SayHello(GaudiAlgo):
    def __init__(self,myname,message="Hello World in Python!"):
        GaudiAlgo.__init__(self,myname)
        self.message = message
        return

    def execute(self):
        print self.message
        return SUCCESS
    pass
```

- Must accept a name as first argument to constructor and pass it up to parent GaudiAlgo class's constructor
- Trivial example, all usual Algorithm features are available.

A nuwa.py aware Python Module driving SayHello

```
my_message = "\nHello World from Python!\n"
def configure(argv = []):
    if len(argv):
        global my_message
        my_message = argv[0]
        print 'Configure with my_message=',my_message
    return

def run(app):
    global my_message
    print 'Making a SayHello with',my_message
    alg = SayHello("Friendly",message=my_message)
    app.addAlgorithm(alg)
    return
```

- Implement `configure()` to access command line arguments.
- In `run()`, create and add algorithm to application.

Writing pure-Python Algorithms, full example

See: `DybHelloWorld/python/DybHelloWorld/PyloWorld.py`
Run: `nuwa.py -n 10 DybHelloWorld.PyloWorld`

Any problems running the example?

Accessing Gaudi Components from Python

From a Python GaudiAlgo, access a service through an interface:

```
def initialize(self):
    sc = GaudiAlgo.initialize(self)
    if sc.isFailure():
        print 'Failed to initialize GaudiAlgo base class'
        return sc
    self.css = self.svc('ICoordSysSvc','CoordSysSvc')
    if not self.css:
        print 'Failed to get CoordSysSvc'
        return FAILURE
    return SUCCESS
```

Then, can call the service's methods in execute().

nuwa.py can run scripts and modules

Running a Job Option Script (JOS)

```
nuwa.py [...] /path/to/some/scripts/my-job-script.py
```

Running a Job Option Module (JOM) with no arguments

```
nuwa.py Package.Module'
```

Running a Job Option Module (JOM) with arguments

```
nuwa.py -m 'Package.Module [options]'
```

Interactive Python with nuwa.py

```
nuwa.py -i -n 1 DybHelloWorld.PyloWorld
(... job runs ...)
In [1]:
```