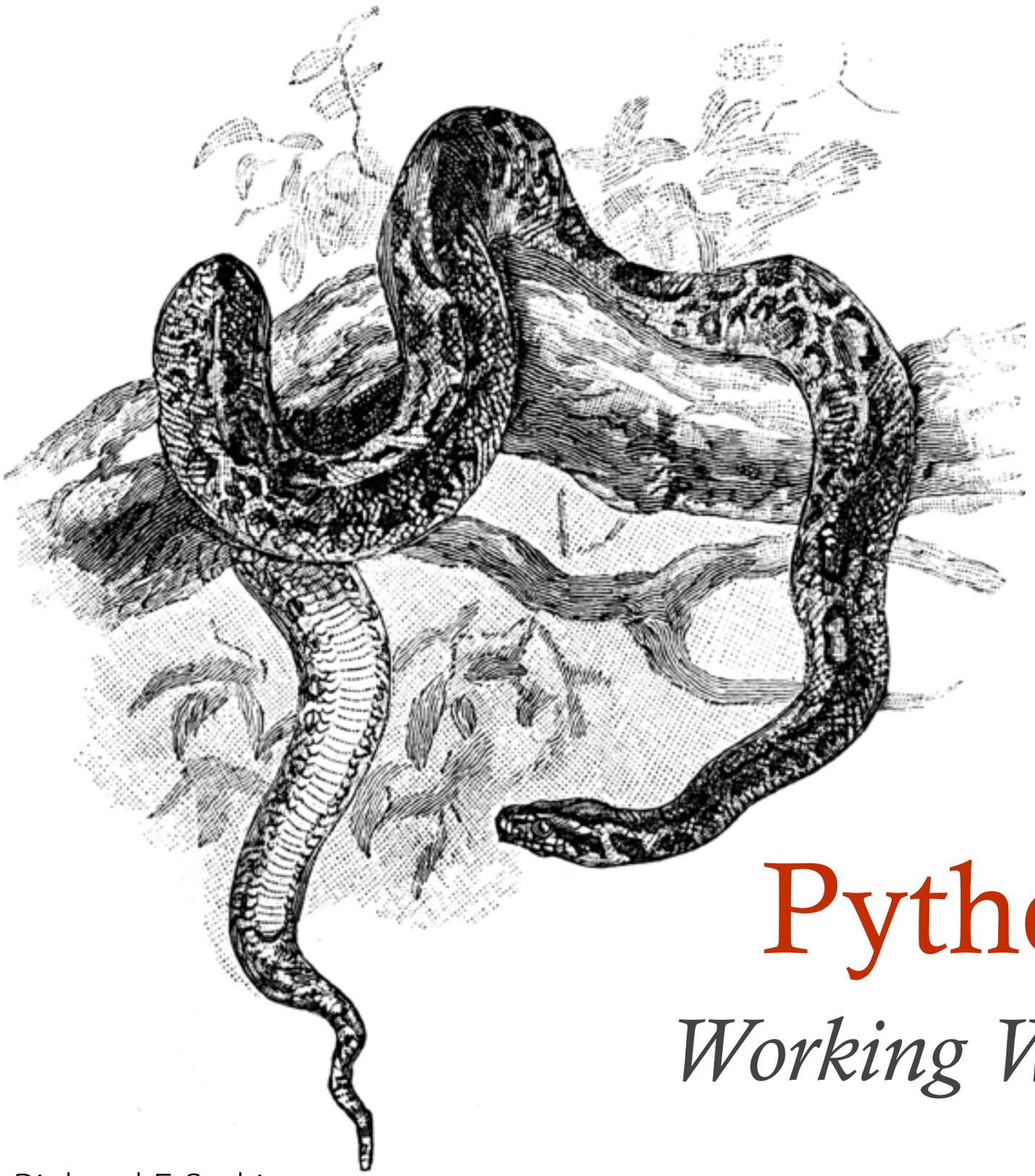


Python Bridges

Working With Other Runtimes

Richard E Sarkis
Rochester's Python User Group
October 21st, 2014 Meeting

Why Extend?

- Because, it can be surprisingly easy!*

- Can allow access to system calls, or external library functions

- Can create entirely new built-in object types

- Performance benefits using a language runtime optimized for parallelism, computation, or messaging paradigms

Let's Ask The Tough Question…

What isn't Python good for?

Python is general purpose

- Not inherently a computational language
- No tail recursion
- No type safety
- Concurrency Shortcomings
- Not great for mobile development
- Not good for embedded applications

Simple Example (Python)

```
total = 0

for i in range(100000000):
    total += i

print total
```

Simple Example (C++)

```c
#include <stdio.h>

int main() {
    volatile long total = 0;

    for (long i = 0; i < 1000000000; i++) {
        total += i;
    }

    printf("%ld\n", total);
}
```

Abstract Example (Python)

```
total = 0


class Adder:
    def __init__(self):
        self.total = 0

    def add(self, i):
        self.total += i

adder = Adder()

for i in range(100000000):
    adder.add(i)

print adder.total
```

Abstract Example (C++)

```
#include <stdio.h>

class Adder {
     public:
          Adder() : total(0) {}
          void add(long i) { total += i; }
          volatile long total;
};

int main() {
     Adder adder;

     for (long i = 0; i < 1000000000; i++) {
          adder.add(i);
     }
    printf("%ld\n", adder.total);
}
```

Python == Jack-Of-All-Trades

C

with special guests,
C++ and Objective-C

CPython API

- Official method of Python extending
- Included with the file **`Python.h`**
- Calling a C library or systems calls? Consider **`ctypes`** instead
    - Can be more portable to other Python implementations of Python, i.e. PyPy
- Useful for embedding Python in a “real application”

The Object Data Type

`PyObject*`

- Most Python/C API functions use this as args and ret values
- Represents an arbitrary Python object
- Live in the heap, not stack; never declared as automatic/static
- All instances have a type and reference count

Coding Guidelines

- **`Python.h`** must be included before standard headers
- **`Py`** or **`PY`** prefixes are user visible from header
  - **`_Py`** prefix is for internal use only
  - Avoid using these prefixes!

A Contrived CPython Extension Module

```
>>> import spam
>>> status = spam.system("ls -l")
```

Simple Example

```
#include <Python.h>
```

Simple Example

```
static PyObject *
spam_system(PyObject *self, PyObject *args)
{
    const char *command;
    int sts;

    if (!PyArg_ParseTuple(args, "s", &command))
        return NULL;
    sts = system(command);

    return Py_BuildValue("i", sts);
}
```

Simple Example

```
Py_INCREF(Py_None);
return Py_None;
```

Simple Example

```
static PyMethodDef SpamMethods[] = {
    ...
    {"system",  spam_system, METH_VARARGS,
     "Execute a shell command."},
    ...
    {NULL, NULL, 0, NULL}        /* Sentinel */
};
```

Simple Example

```
static PyObject *SpamError;

PyMODINIT_FUNC
initspam(void)
{
    PyObject *m;

    m = Py_InitModule("spam", SpamMethods);
    if (m == NULL)
        return;

    SpamError = PyErr_NewException("spam.error", NULL, NULL);
    Py_INCREF(SpamError);
    PyModule_AddObject(m, "error", SpamError);
}
```

Simple Example

```
static PyObject *
spam_system(PyObject *self, PyObject *args)
{
    const char *command;
    int sts;

    if (!PyArg_ParseTuple(args, "s", &command))
        return NULL;
    sts = system(command);
    if (sts < 0) {
        PyErr_SetString(SpamError, "System command failed");
        return NULL;
    }
    return PyLong_FromLong(sts);
}
```

Simple Example: Hand Compile

```
# Compile
gcc `/usr/local/bin/python-config --cflags` spammodule.c -c —o spammodule.o

# Link
gcc -shared `/usr/local/bin/python-config --ldflags` spammodule.o -o spam.so
```

Simple Example: setuptools

```
# setup.py
from distutils.core import setup, Extension

spam_module = Extension('spam', sources=['spammodule.c'])

setup(name='SpamModule',
      version='1.0',
      description='This is a spammy package',
      ext_modules=[spam_module])
```

Simple Example

```
python setup.py install
```

Demo

References

- Extending Python with C or C++
  https://docs.python.org/2/extending/extending.html
- Building C and C++ Extensions with distutils
  https://docs.python.org/2/extending/building.html
- Python/C API Reference Manual
  https://docs.python.org/2/c-api/intro.html

A Dumb Example of Calling Python from C

Pyception

- We can make a C program that runs an internal Python interpreter
- We can load modules, including our `spam` module
- We can pass it C strings containing Python code

Demo

Calling Objective-C from Python

Foreign Exchange

- CFFI, a Foreign Function Interface for Python calling C code is available for CPython and PyPy
- Enables you to call existing C libraries, rather than compiling external C modules for embedding (importing) into Python
- One very clean example of using CFFI is NSPython

NSPython

- "NSPython is a simple Python library for using Objective C, Foundation and Application Kit Frameworks, also known as Cocoa."
- Cocoa is written almost entirely in Objective-C, which is a true superset of the C language

Objective-C Example

```
@"hello"
@selector(setDelegate:)

[myObject delegate];
[myObject setVariable:YES anotherVariable:NO];

[NSString stringWithUTF8String:"hello"];
[[NSString alloc] initWithUTF8String:"hello"] autorelease];

[super init];
```

NSPython

```
at('hello')
sel('setDelegate:')

myObject.delegate()
myObject.setVariable_anotherVariable_(True, False)

NSString.stringWithUTF8String_('hello')
NSString.alloc().initWithUTF8String_('hello').autorelease()

get_super(self).init()
```

Subclassing

```
class MyString(NSString):

    def initWithUTF8String_(self, string): pass

    @types('id', 'char *')
    @classmethod
    def anotherStringWithUTF8String_(self, string): pass

    @types('id', 'char *')
    def anotherInitWithUTF8String_(self, string): pass
```

Demo

References

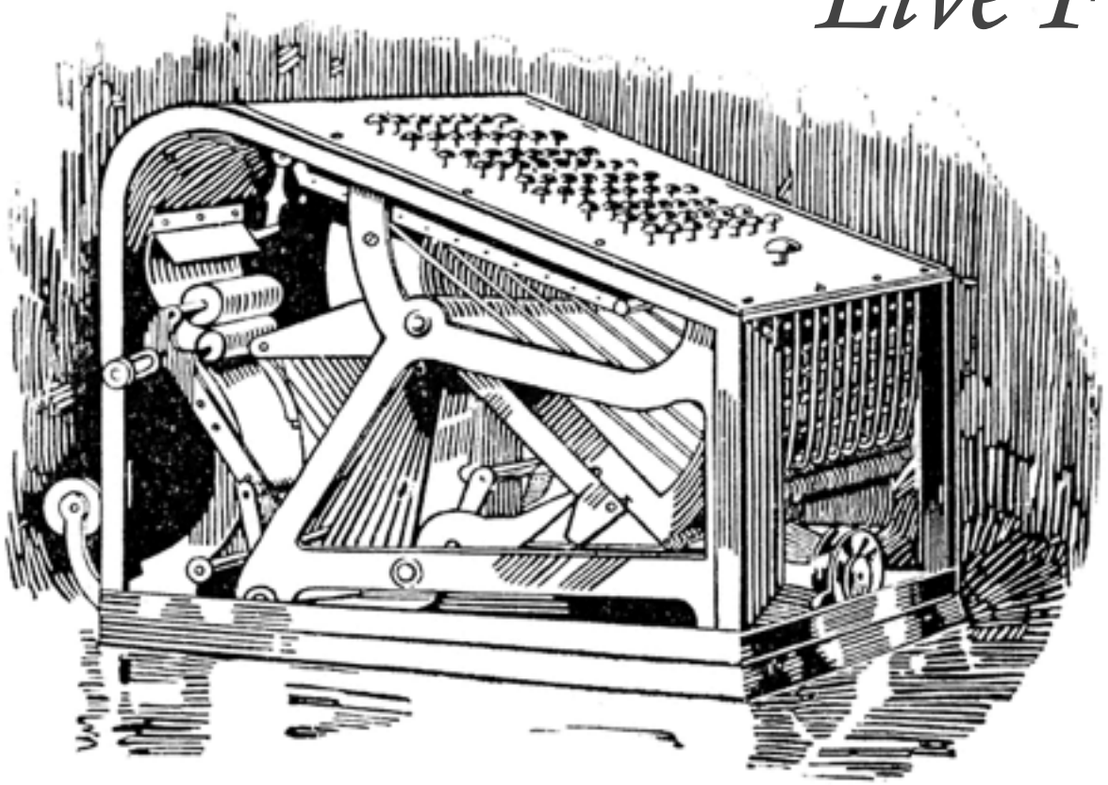
- NSPython
  https://bitbucket.org/sukop/nspython

http://www.ucs.cam.ac.uk/docs/course-notes/unix-courses/pythonfortran/files/f2py.pdf

https://sysbio.ioc.ee/projects/f2py2e/

Fortran

Live Free or π Hard



Why Fortran

- Pure Python can't compete with speed
- Fortran is a natively compiled language
- Designed for numerical work (**For**mula **Tran**slation)
- 50+ year history
- We can get the best of both worlds!
  - Python: high-level code, glue
  - Fortran: heavy crunching

Pure Python

- Main Program: `program.py`
- Module: `thing.py`
- Run with: `time python program.py`

Mixed Python/Fortran

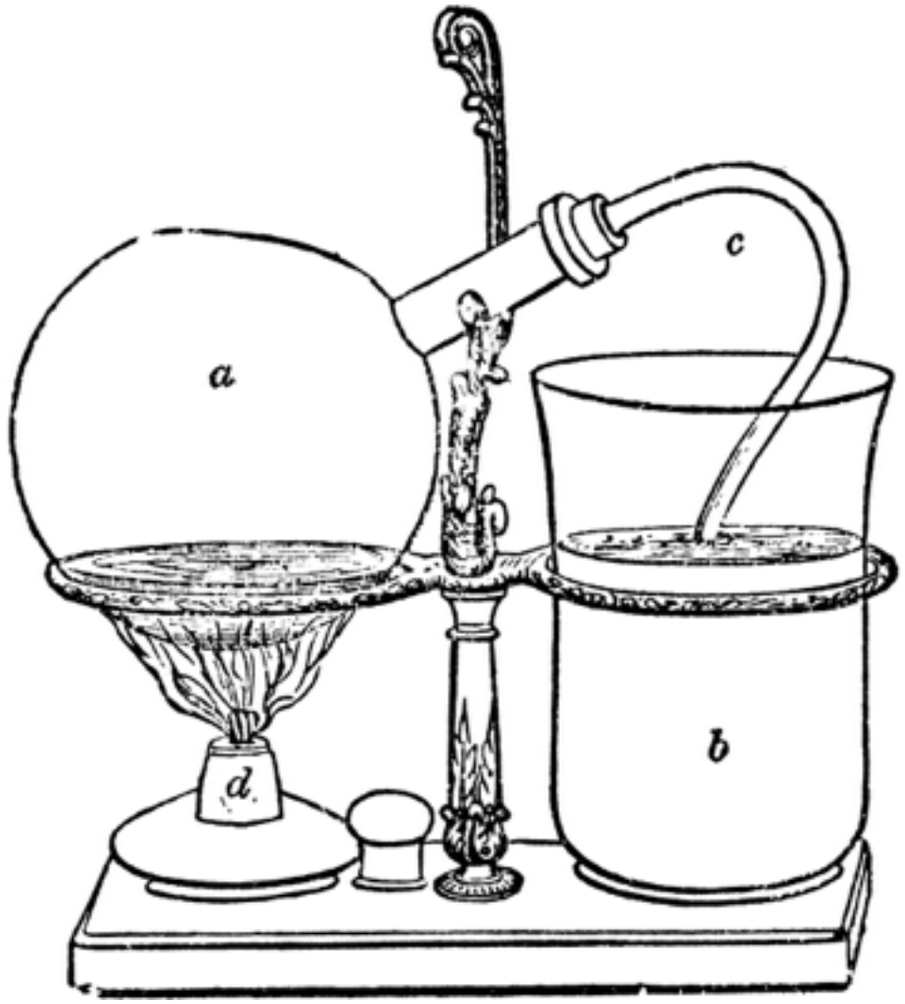
- Main Program: `program.py`
- Module: `thing.f95`
- Run with: `time python program.py`

Demo

References

- **Interfacing Python with Fortran**
  http://www.ucs.cam.ac.uk/docs/course-notes/unix-courses/pythonfortran/files/f2py.pdf

https://wiki.python.org/moin/Jython



Java

A Special Blend

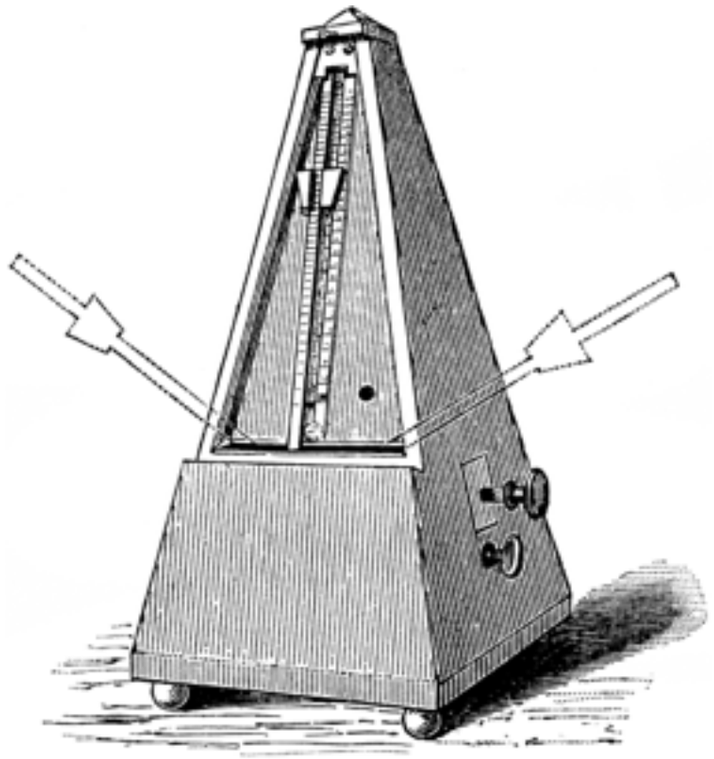
Jython

- Jython is a Python interpreter implemented in Java
- Integrates into existing Java applications
- Can compile Python into Java classes
- Python in Jython has access to Java classes & APIs

References

- Jython
  http://www.jython.org/

http://www.codeplex.com/IronPython



C#

IronPython

- IronPython is a Python interpreter implemented in C#
- Integrates into .NET and Mono
- Runs using Dynamic Language Runtime (DLR), a library running on top of the Common Language Infrastructure that provides dynamic typing and dynamic method dispatch

References

- IronPython
  http://ironpython.net/

Conclusions

- Python can be implemented 6-ways from Sunday
- Compiled languages provide the potential for speed benefit, when integrated into Python
- Alternative Python implementations can provide the Python language for a different language runtime/VM
- https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages?

Questions?