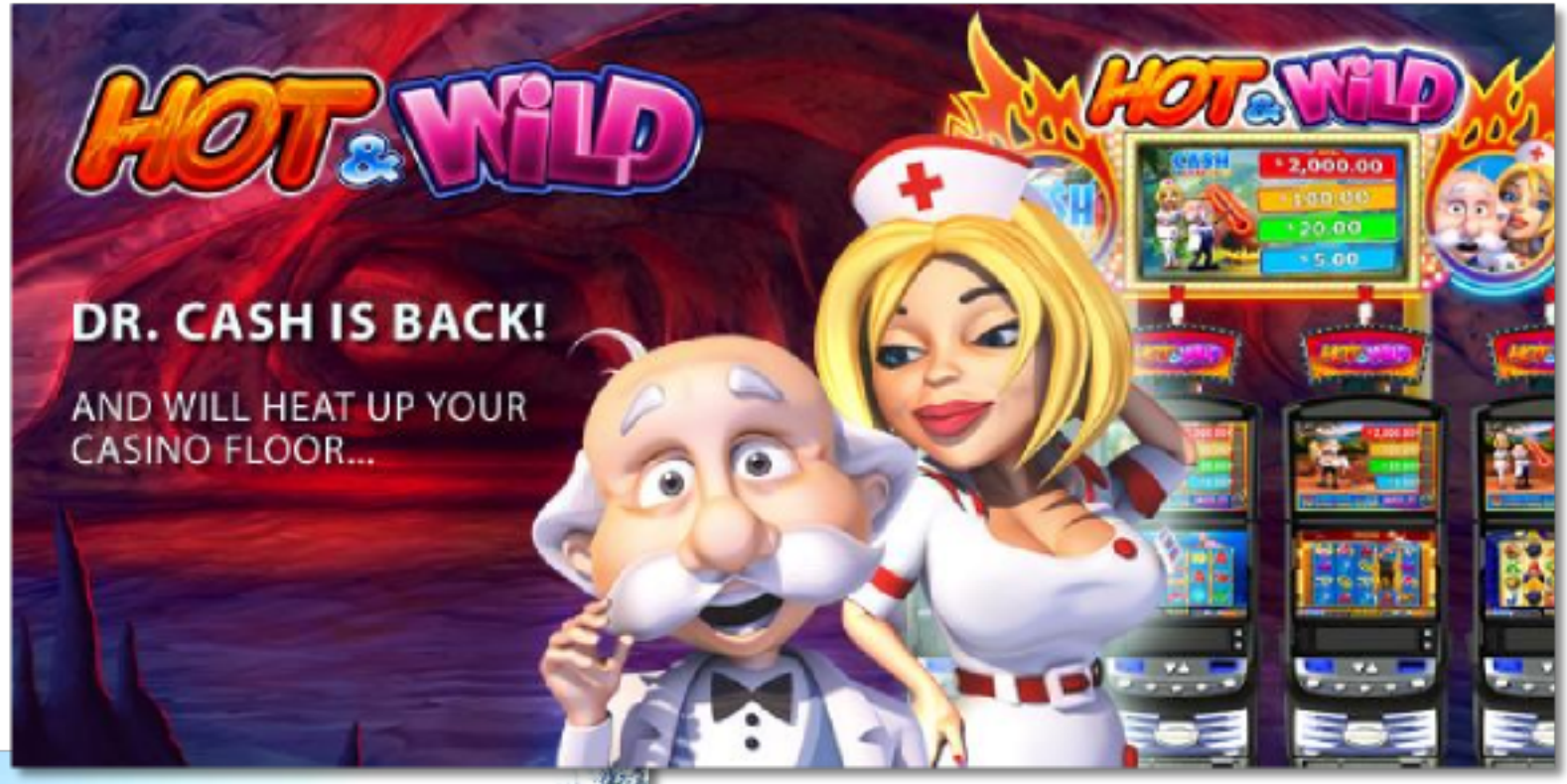
Supercharging C++ Code with Embedded Python

EuroPython 2012

Michael Fötsch | @mfoetsch | realmike.org

SPIELO International

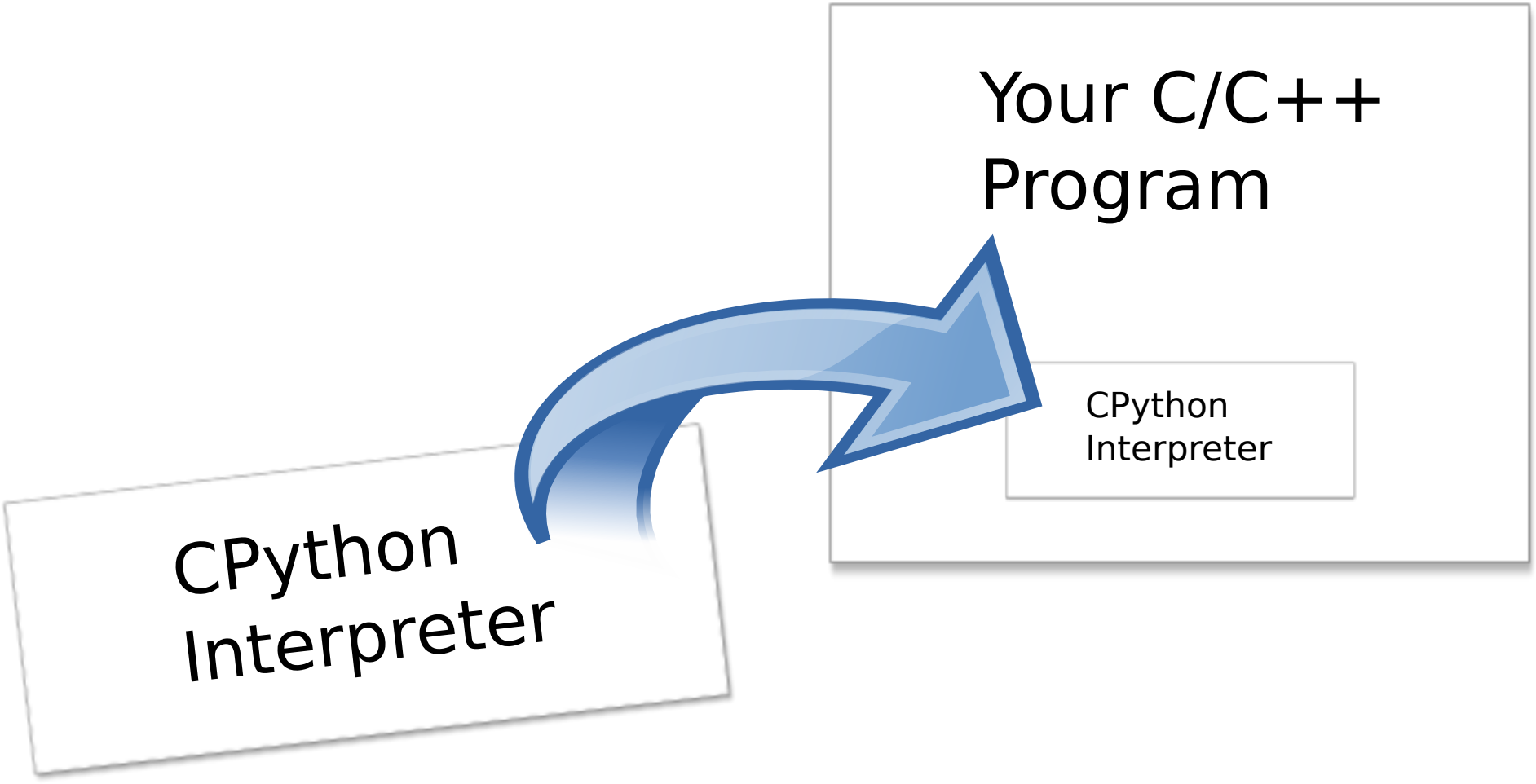


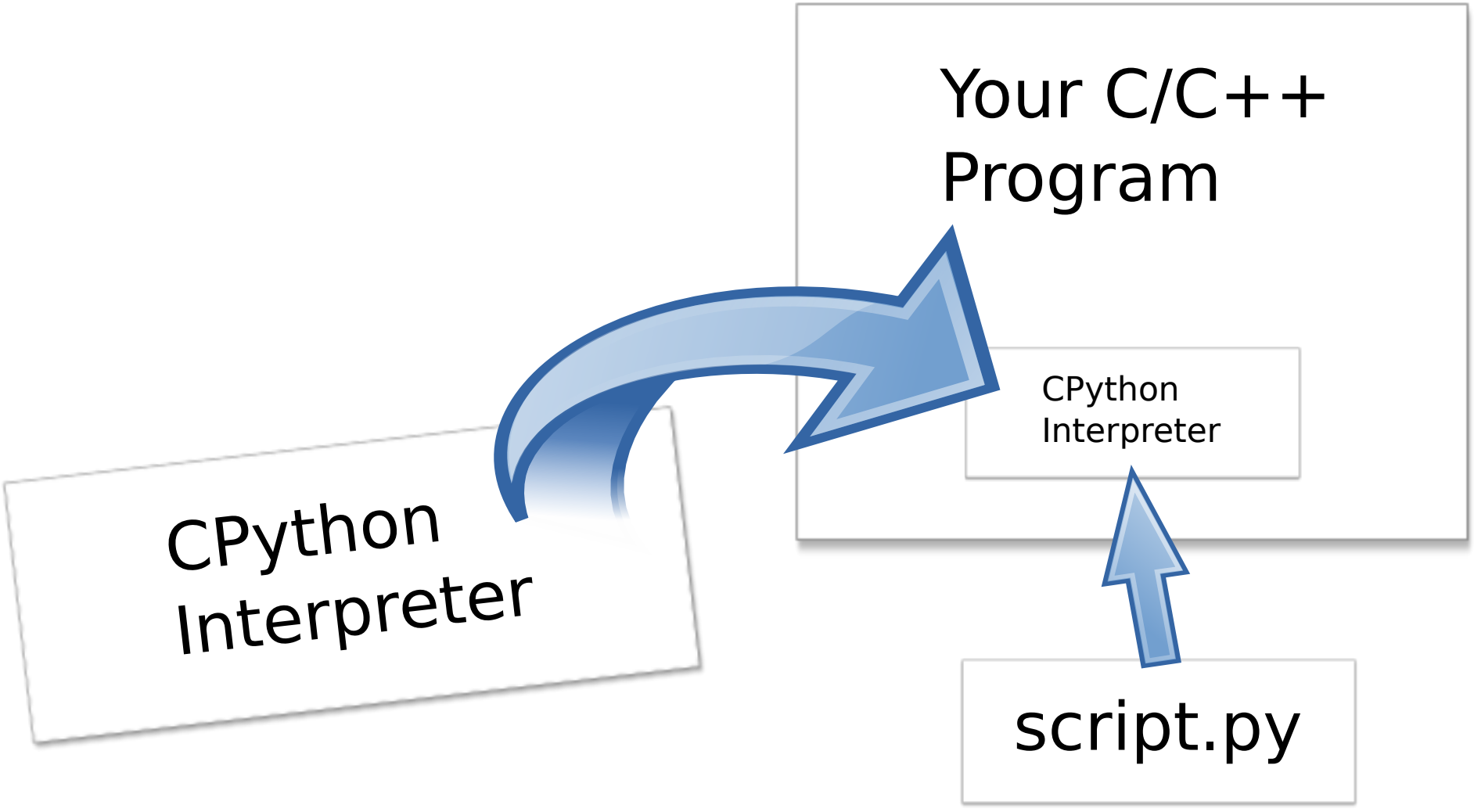
spielo.com/careers

❶ Use cases.

❷ From extending to embedding.

❸ Case study.





macro.py

```
import random
for word in currentDocument.allWords:
    style = random.choice(allStyles)
    word.applyStyle(style)
```

test.py

```
mechButtons["PLAY"].press()
assert reels.spinning
time.sleep(2)
assert not reels.spinning
```

```
class Guard(Enemy):
    def OnGettingHit(self, actor):
        self.findCover()
        self.shootAt(actor)
        self.team.setAlarmed(True)
```

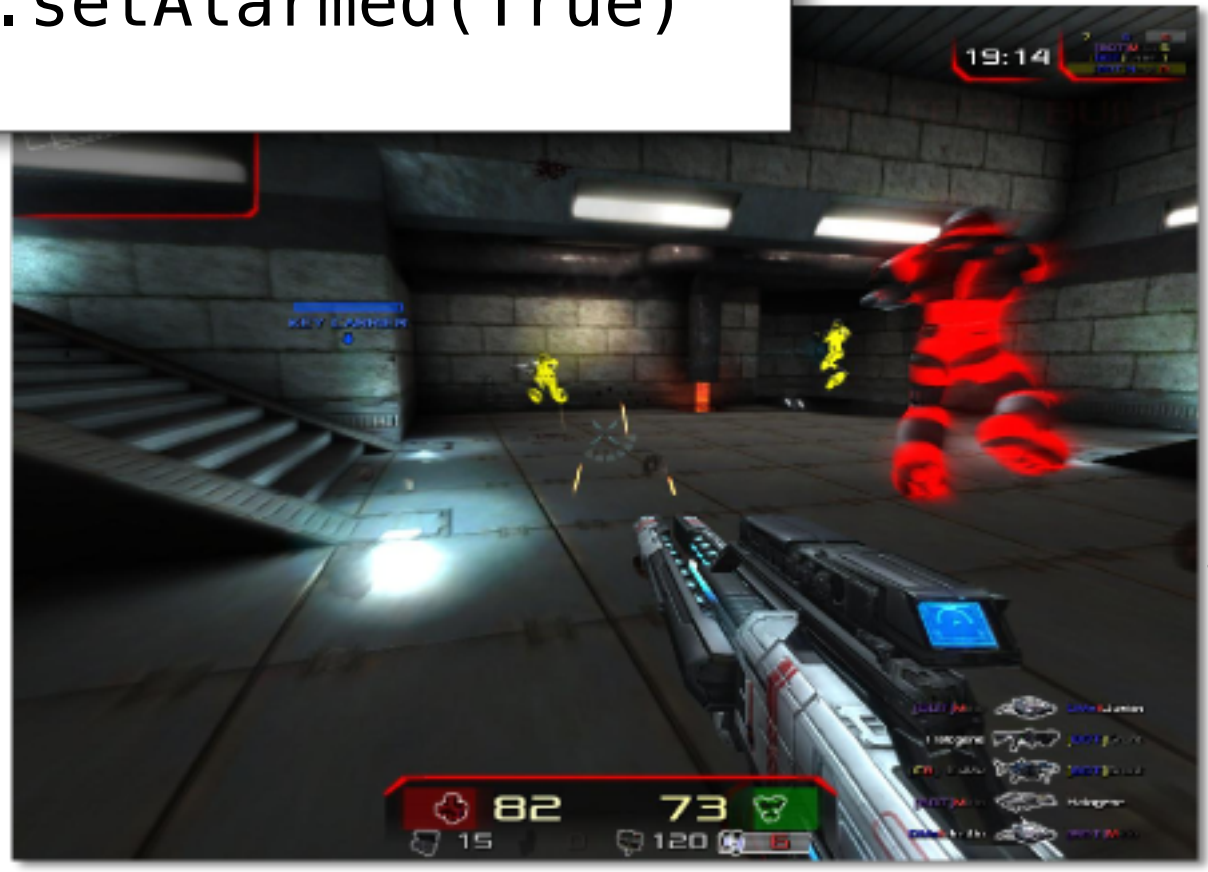


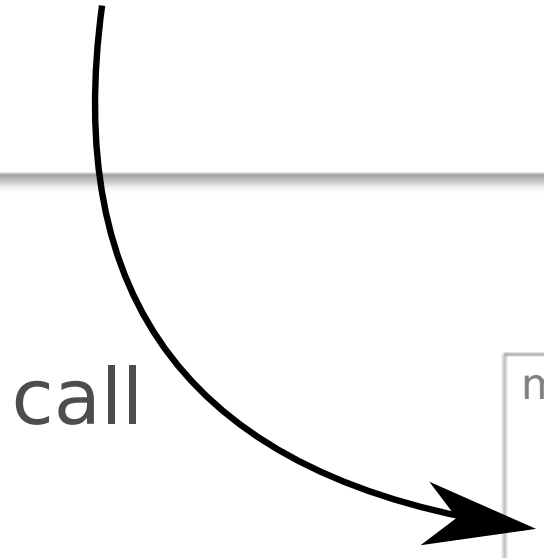
Image source: xonotic.org

- Ease of use.
- Sandboxing.
- Flexibility.

Extending Recap.

py_program.py

```
import my_c_lib as c
print c.Sum(5, 3.2)
```



my_c_lib.cpp

```
int Sum(int a, int b)
{
    return a + b;
}
```

py_program.py

```
LOAD_GLOBAL    0 (c)
LOAD_ATTR      1 (Sum)
LOAD_CONST     1 (5)
LOAD_CONST     2 (3.2)
CALL_FUNCTION 2
PRINT_ITEM
PRINT_NEWLINE
```

my_c_lib.cpp

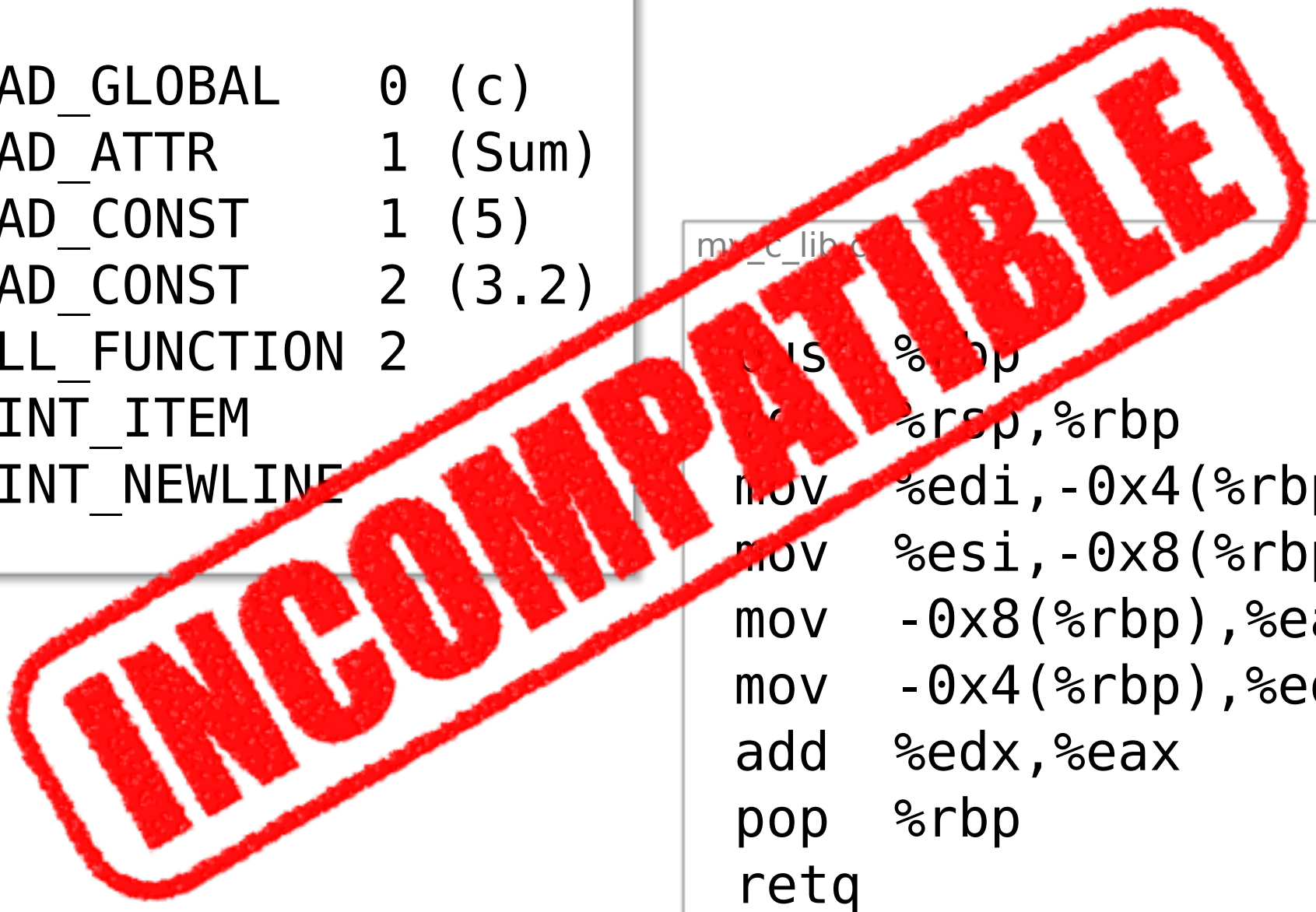
```
push %rbp
mov  %rsp,%rbp
mov  %edi,-0x4(%rbp)
mov  %esi,-0x8(%rbp)
mov  -0x8(%rbp),%eax
mov  -0x4(%rbp),%edx
add  %edx,%eax
pop  %rbp
retq
```

py_program.py

```
LOAD_GLOBAL    0 (c)
LOAD_ATTR      1 (Sum)
LOAD_CONST     1 (5)
LOAD_CONST     2 (3.2)
CALL_FUNCTION  2
PRINT_ITEM
PRINT_NEWLINE
```

my_c_lib.c

```
push   %rbp
mov    %rsp,%rbp
mov    %edi,-0x4(%rbp)
mov    %esi,-0x8(%rbp)
mov    -0x8(%rbp),%eax
mov    -0x4(%rbp),%edx
add    %edx,%eax
pop    %rbp
retq
```



Python

```
x = 5
```

```
{ob_refcnt = 18
 ob_type = 0x7cd7060
 ob_ival = 5}
```

C++

```
int x = 5;
```

```
0x00000005
```

Python

```
x = 5
```

```
{ob_refcnt = 18
 ob_type = 0x7cd7060
 ob_ival = 5}
```


C++

```
int x = 5;
```



```
0x00000005
```

mathmodule.c

```
static PyObject* math_radians(
    PyObject* self, PyObject* arg)
{
    double x = PyFloat_AsDouble(arg);
    return PyFloat_FromDouble(
                        x * PI / 180.0);
}
```

my_c_lib.cpp

```
int Sum(int a, int b)
{
    return a + b;
}
```

my_c_lib.cpp

```
static PyObject* WrapSum(
  PyObject* self, PyObject* args)
{









}
```

my_c_lib.cpp

```
static PyObject* WrapSum(
  PyObject* self, PyObject* args)
{   PyObject* oa;
    PyObject* ob;
    PyArg_UnpackTuple(
        args, "Sum", 2, 2, &oa, &ob);




}
```

my_c_lib.cpp

```
static PyObject* WrapSum(
  PyObject* self, PyObject* args)
{   PyObject* oa;
    PyObject* ob;
    PyArg_UnpackTuple(
        args, "Sum", 2, 2, &oa, &ob);
    long a = PyInt_AsLong(oa);
    long b = PyInt_AsLong(ob);




}
```

my_c_lib.cpp

```
static PyObject* WrapSum(
  PyObject* self, PyObject* args)
{   PyObject* oa;
    PyObject* ob;
    PyArg_UnpackTuple(
        args, "Sum", 2, 2, &oa, &ob);
    long a = PyInt_AsLong(oa);
    long b = PyInt_AsLong(ob);
    // call the original Sum()
    long result = Sum(a, b);

}
```

my_c_lib.cpp

```
static PyObject* WrapSum(
  PyObject* self, PyObject* args)
{   PyObject* oa;
    PyObject* ob;
    PyArg_UnpackTuple(
        args, "Sum", 2, 2, &oa, &ob);
    long a = PyInt_AsLong(oa);
    long b = PyInt_AsLong(ob);
    // call the original Sum()
    long result = Sum(a, b);
    return PyInt_FromLong(result);
}
```

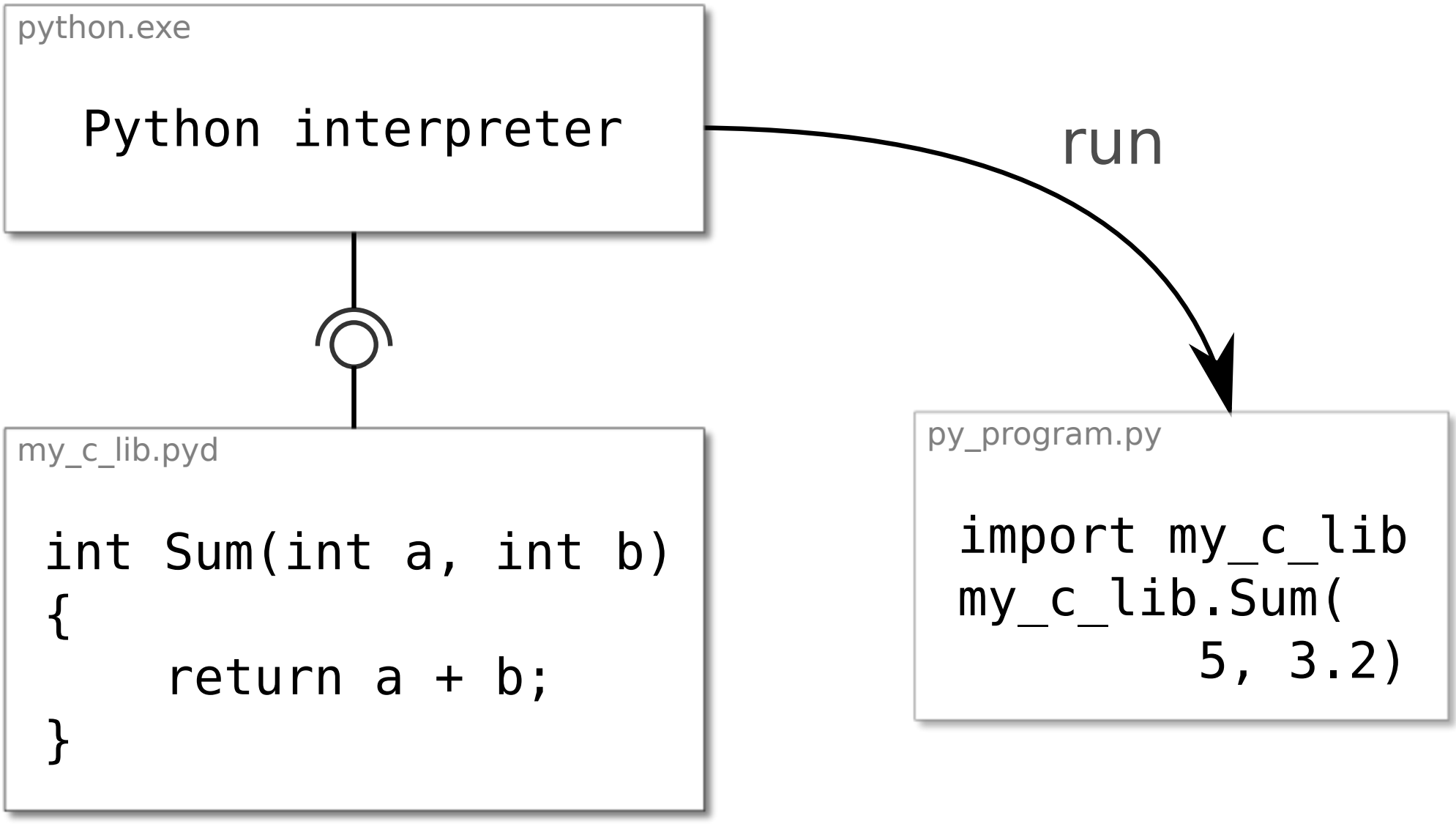
my_c_lib.cpp

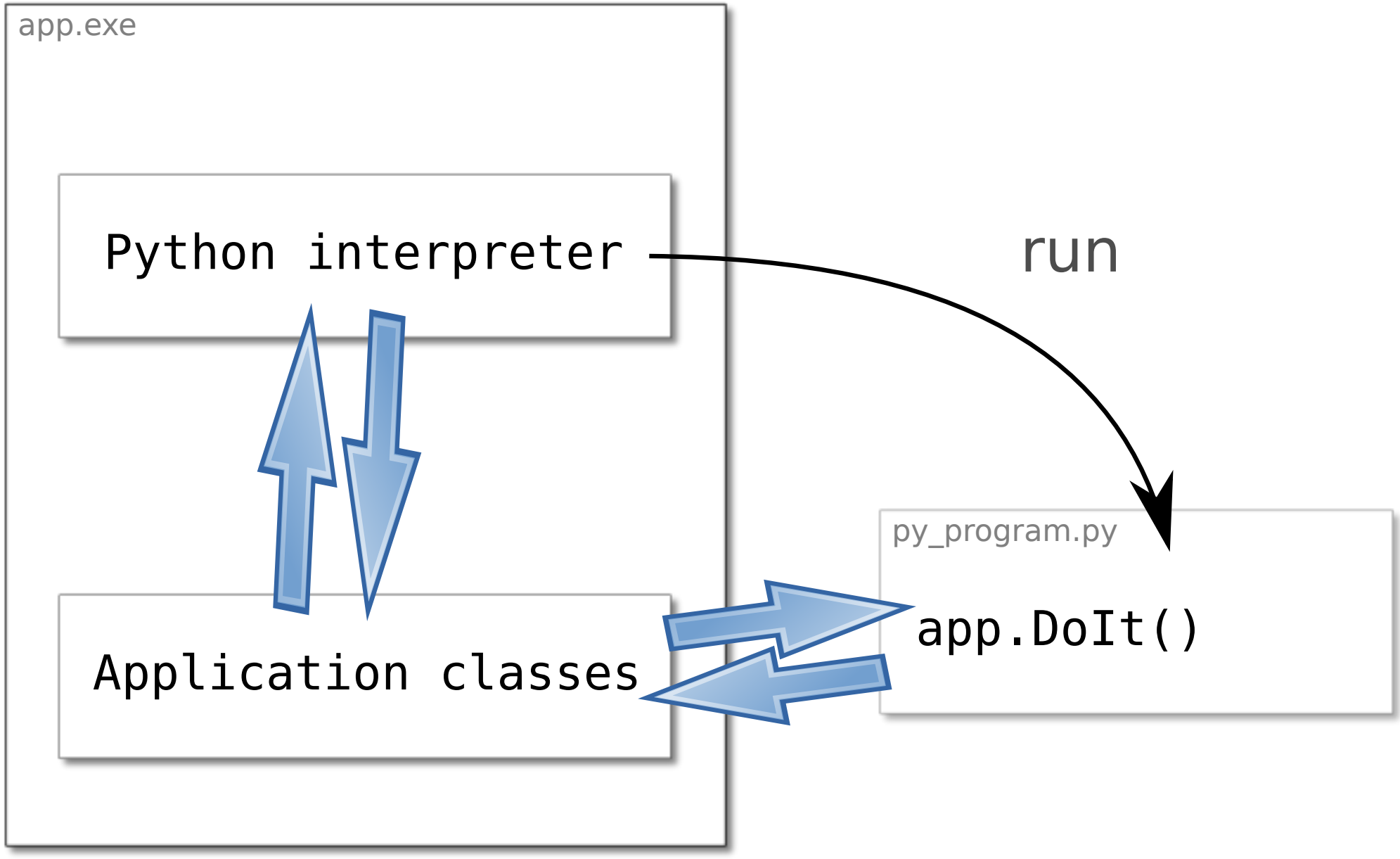
```
static PyMethodDef MyLibMethods[] =
{   {"Sum", WrapSum, METH_VARARGS,
     "Calculate sum of two ints."},
    {NULL, NULL, 0, NULL}
};

PyMODINIT_FUNC initmy_c_lib(void)
{
   (void)Py_InitModule("my_c_lib",
                       MyLibMethods);
}
```

- Extension module = DLL/shared object.
- Take `PyObjects`, return `PyObjects`.
- Python/C API to convert.

- SWIG. Simplified Wrapper Interface Generator.
- Boost.Python.

From Extending to Embedding.





high_level.cpp

```
#include <Python.h>
int main(int argc, char* argv[])
{
    Py_Initialize();
    PyRun_SimpleString(
        "name = raw_input('Name: ')\n"
        "print 'Hi, %s!' % name\n");
    Py_Finalize();
    return 0;
}
```

program.cpp

```
while (true)
{    string input, output;
     getline(cin, input);
     output = CallPythonFilter(input);
     cout << input << endl;
}
```

filter.py

```
def filterFunc(s):
    return ???
```

elmer_fudd_filter.py

```
def filterFunc(s):
    return s.replace(
        "r", "w").replace("l", "w")
```

shout_filter.py

```
def filterFunc(s):
    return s.upper()
```

```
string CallPythonFilter(string& s)
{



}
```

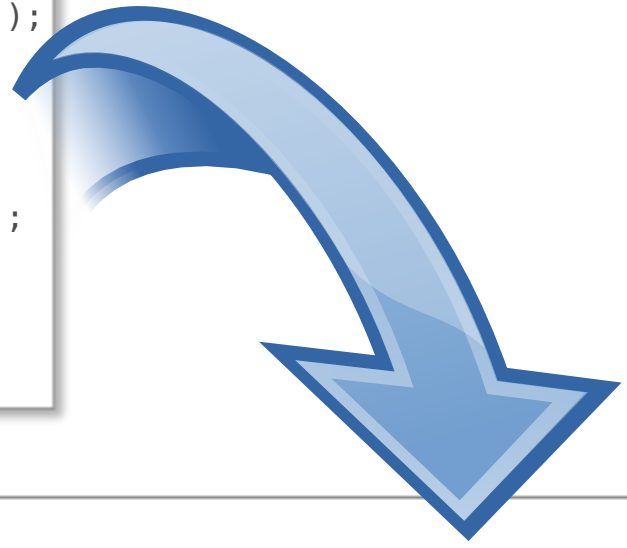
```
string CallPythonFilter(string& s)
{ PyObject* pluginModule
    = PyImport_Import(
        PyString_FromString("filter"));









}
```

```
string CallPythonFilter(string& s)
{ PyObject* pluginModule
    = PyImport_Import(
        PyString_FromString("filter"));
  PyObject* filterFunc
    = PyObject_GetAttrString(
        pluginModule, "filterFunc");




}
```

```
string CallPythonFilter(string& s)
{ PyObject* pluginModule
    = PyImport_Import(
        PyString_FromString("filter"));
  PyObject* filterFunc
    = PyObject_GetAttrString(
        pluginModule, "filterFunc");
  PyObject* argsTuple
    = Py_BuildValue("(s)", s.c_str());



}
```

```
string CallPythonFilter(string& s)
{ PyObject* pluginModule
    = PyImport_Import(
        PyString_FromString("filter"));
  PyObject* filterFunc
    = PyObject_GetAttrString(
        pluginModule, "filterFunc");
  PyObject* argsTuple
    = Py_BuildValue("(s)", s.c_str());
  return PyString_AsString(
    PyObject_CallObject(
      filterFunc, argsTuple));
}
```

```
string CallPythonFilter(string& s)
{ PyObject* pluginModule
    = PyImport_Import(
        PyString_FromString("filter"));
  PyObject* filterFunc
    = PyObject_GetAttrString(
        pluginModule, "filterFunc");
  PyObject* argsTuple
    = Py_BuildValue("(s)", s.c_str());
  return PyString_AsString(
    PyObject_CallObject(
      filterFunc, argsTuple));
}
```



```
def CallPythonFilter(s):
    pluginModule = __import__("filter")
    filterFunc = getattr(
            pluginModule, "filterFunc")
    argsTuple = (s,)
    return filterFunc(*args)
```

- Python code $\Longrightarrow$ Python/C API calls.
- Convert `PyObjects` $\Longleftrightarrow$ C++.
- Wrap C++ objects.

- Boost.Python.
- PyCXX.

Familiarity.

Maturity.

Experience.

Why Python?

Community.

Portability.

Licensing.

Use the **fork**, Luke.

Sandboxing.

Debugging.

Image source: http://www.flickr.com/photos/juan_e/. License: CC-BY-SA 2.0

More embedding: realmike.org

We're hiring: spielo.com/careers