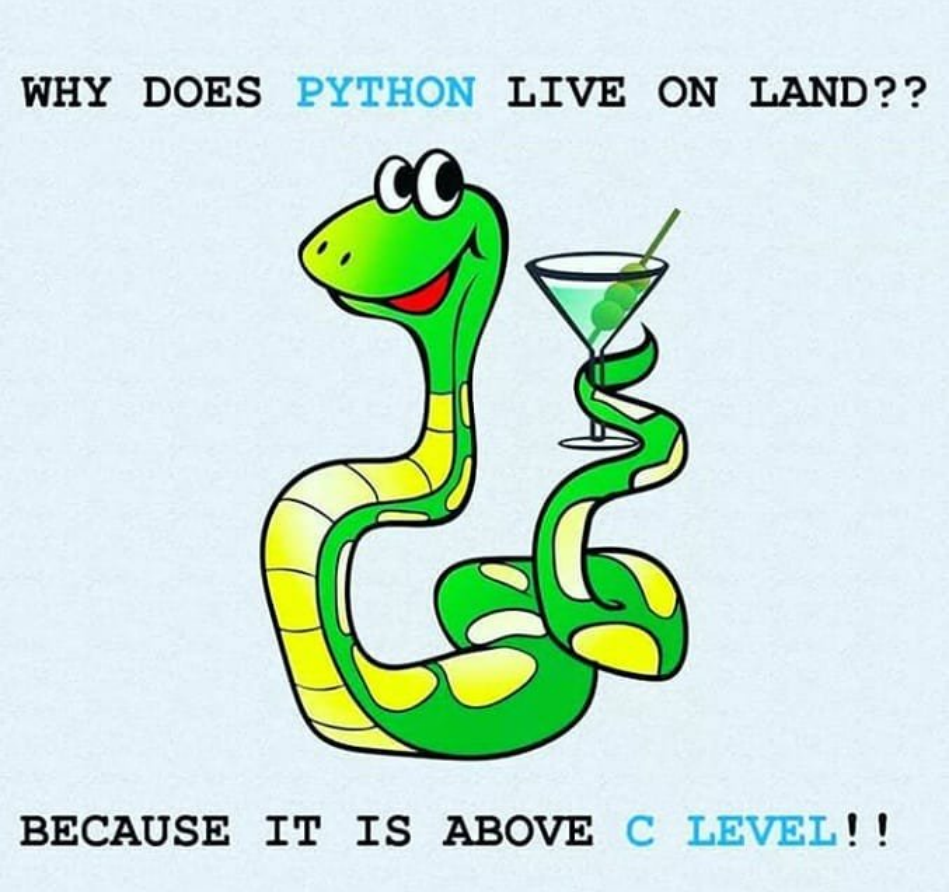
Bindings & Extensions



Problems:

- You have lots of useful C/C++ code
- You want use C++ in your app bottleneck

Extensions

1. Take C++ code
2. Define interaction between C++ and Py
3. Compile it as shared library
4. Use in Py code

```cpp
#include <string>

class EmptyArgumentException : public std::runtime_error {
    using std::runtime_error::runtime_error;
};

std::string cpp_concat(const std::string& l, const std::string& r) {
    if (l.empty() || r.empty()) {
        throw EmptyArgumentException{"Empty argument passed"};
    }
    return l + r;
}
```

Describe py function

```
static PyObject* py_concat(PyObject* /* self */, PyObject* args) {
    PyObject* result = NULL;
    return result;
}
```

Describe py function

```
static PyObject* py_concat(PyObject* /* self */, PyObject* args) {
    PyObject* result = NULL;
    const char *l, *r;
    if (!PyArg_ParseTuple(args, "ss", &l, &r)) {
        return NULL;
    }
    return result;
}
```

Describe py function

```
static PyObject* py_concat(PyObject* /* self */, PyObject* args) {
    PyObject* result = NULL;
    const char *l, *r;
    if (!PyArg_ParseTuple(args, "ss", &l, &r)) {
        return NULL;
    }

    auto concatenated = cpp_concat({l}, {r});
    result = Py_BuildValue("s", concatenated.data());
    if (result == NULL) {
        return result;
    }
    return result;
}
```

Describe py function

```
static PyObject* py_concat(PyObject* /* self */, PyObject* args) {
    PyObject* result = NULL;
    const char *l, *r;
    if (!PyArg_ParseTuple(args, "ss", &l, &r)) {
        return NULL;
    }

    try {
        auto concatenated = cpp_concat({l}, {r});
        result = Py_BuildValue("s", concatenated.data());
        if (result == NULL) {
            return result;
        }
    } catch (const EmptyArgumentException& e) {
        PyErr_SetString(PyExc_RuntimeError, e.what());
    }
    return result;
}
```

Describe py module

```
static PyMethodDef methods[] = {
    {"concat", py_concat, METH_VARARGS, "Concat 2 non-empty strings"},
    {NULL, NULL, 0, NULL}        /* Sentinel */
};


static struct PyModuleDef module = {
    PyModuleDef_HEAD_INIT,
    "concat module",    /* module name */
    NULL,               /* documentation, may be NULL */
    -1,                 /* size of per-interpreter module state,
                         * -1 if the module keeps state in globals
                         */
    methods
};
```

Describe py module

```
// Note: name `_libconcat` will be the name of your module
PyMODINIT_FUNC
PyInit_libconcat() {
    return PyModule_Create(&module);
}
```

Compile

- use python3-config to get flags python was compiled with

```
g++ $(python3-config --cflags --ldflags) -shared --std=c++17 \
concat.cpp -o libconcat.so
```

In [2]:

```
import libconcat

print(libconcat.__name__)

l = 'a' * 10000
r = 'b' * 10000
res = libconcat.concat(l, r)
print(len(res))

libconcat.concat('', r)
```

```
concat module
20000

---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-2-c7d25fcca77e> in <module>
      8 print(len(res))
      9
---> 10 libconcat.concat('', r)

RuntimeError: Empty argument passed
```

Anything about memory management?

```
static PyObject* py_concat_list(PyObject* self, PyObject* args) {
    PyObject* list;
    if (!PyArg_ParseTuple(args, "O", &list)) {
        return NULL;
    }
    int n = PyList_Size(list);
    if (n < 2) {
        Py_DECREF(list);
        PyErr_SetString(PyExc_RuntimeError, "Sequence of len >= 2 required");
        return NULL;
    }
    for (int i = 0; i < n; ++i) {
        /*
         * process i-th item somehow
         */
    }
    Py_DECREF(list);
    return result;
}
```

What's inside loop

```
PyObject *item = NULL, *result = NULL;

for (int i = 0; i < n; ++i) {
    item = PyList_GetItem(list, i);
    if (result == NULL) {
        result = item;
        continue;
    }

    PyObject* subcall_args = Py_BuildValue("OO", result, item);
    PyObject* subcall_res = py_concat(self, subcall_args);
    if (subcall_res == NULL) {
        return NULL;
    }
    result = subcall_res;
}
```

What's inside loop

```
PyObject *item = NULL, *result = NULL;
for (int i = 0; i < n; ++i) {
    item = PyList_GetItem(list, i);
    if (result == NULL) {
        result = item;
        Py_INCREF(result);
        continue;
    }
    PyObject* subcall_args = Py_BuildValue("OO", result, item);
    PyObject* subcall_res = py_concat(self, subcall_args);
    Py_DECREF(subcall_args);
    Py_XDECREF(result);
    if (subcall_res == NULL) {
        Py_DECREF(list);
        return NULL;
    }
    result = subcall_res;
}
```

In [3]:

```
from sys import getrefcount

l = 'a' * 10000
r = 'b' * 10000

def print_refcounts(str):
    print('{}: \tl-refcount = {}, r-refcount = {}'.format(str, getrefcount(l), get
refcount(r)))

print_refcounts('before')
args_list = [l, r, l]
print_refcounts('list created')
res = libconcat.concat_list([l, r, l])
print_refcounts('after call')
del(args_list)
print_refcounts('list deleted')
```

```
before:         l-refcount = 2, r-refcount = 2
list created:   l-refcount = 4, r-refcount = 3
after call:     l-refcount = 4, r-refcount = 3
list deleted:   l-refcount = 2, r-refcount = 2
```

Too much boilerplate?

Cython (static compiler)

```
from libcpp.string cimport string

cdef extern from "concat.h":
    string cpp_concat(string l, string r) except +


def concat(str l, str r):
    cdef string res = cpp_concat(l.encode('utf-8'), r.encode('utf-8'))
    return res.decode('utf-8')

def concat_list(list args):
    if len(args) < 2:
        raise RuntimeError("Expected at least 2 arguments")

    cdef string res = args[0].encode('utf-8')
    for i in range(1, len(args)):
        res = cpp_concat(res, args[i].encode('utf-8'))
    return res.decode('utf-8')
```

Compile

```
# cyconcat.pyx => cyconcat.c
cython cyconcat.pyx

# cyconcat.c => cyconcat.so
g++ $(python3-config --cflags --ldflags) -shared --std=c++17 cyconcat.c -o cyconc
at.so
```

OR

```
# cyconcat.pyx => cyconcat.so
cythonize -i libconcat_cy.pyx
```

In [5]:

```
import cyconcat

print(cyconcat.__name__)
l = 'a' * 10000
r = 'b' * 10000
res = cyconcat.concat(l, r)
print(len(res))

cyconcat.concat('', r)
```

```
cyconcat
20000

---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-5-4d314ea3aa49> in <module>
      7 print(len(res))
      8
----> 9 cyconcat.concat('', r)

~/YSDA_python/cyconcat.pyx in cyconcat.concat()
      6
      7 def concat(str l, str r):
----> 8     cdef string res = cpp_concat(l.encode('utf-8'), r.encode('utf-8'))
      9     return res.decode('utf-8')
     10

RuntimeError: Empty argument passed
```