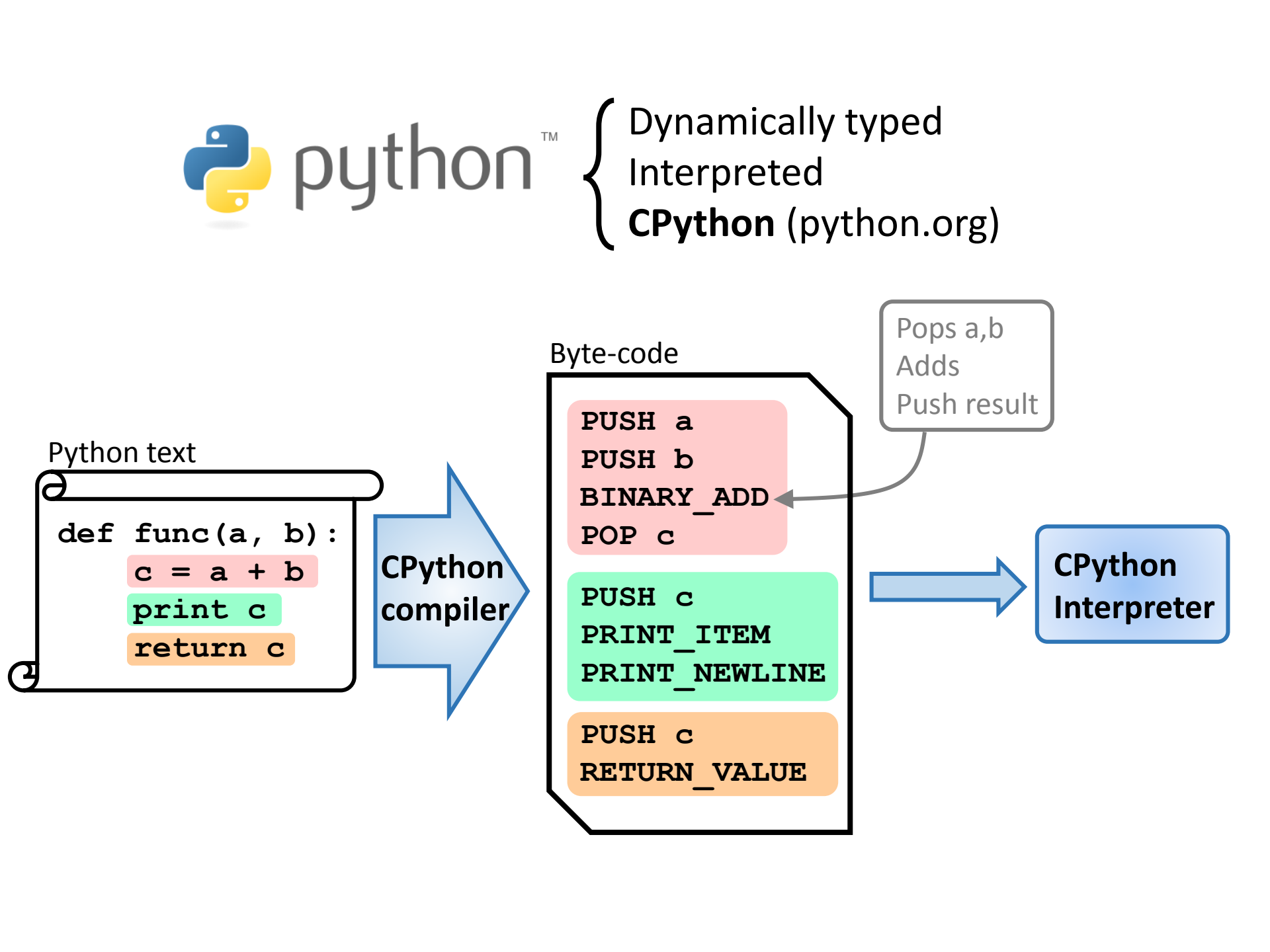
Lightning Talk: Writing a Python Interpreter for Fun and Profit

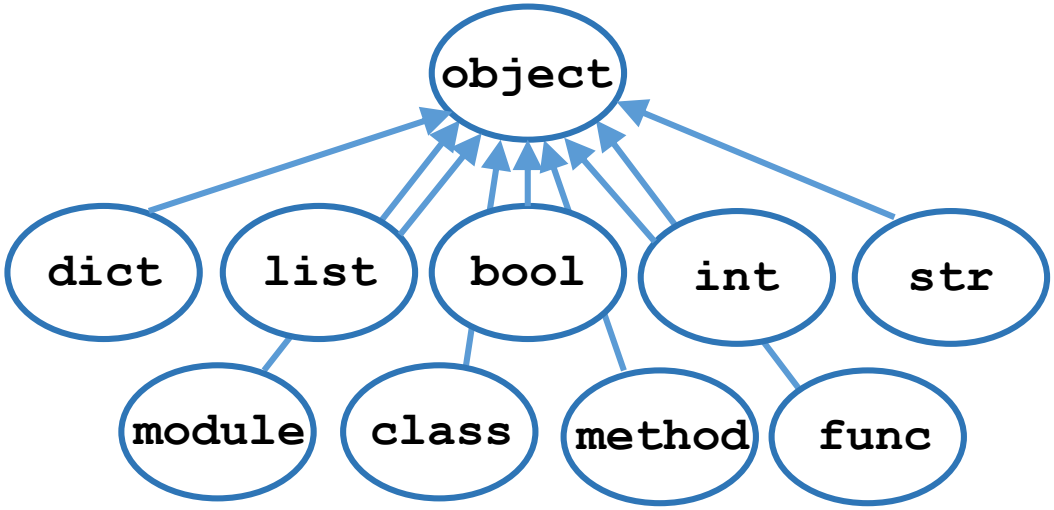
Shy Shalom

Intigua Inc., Israel

shooshx@gmail.com

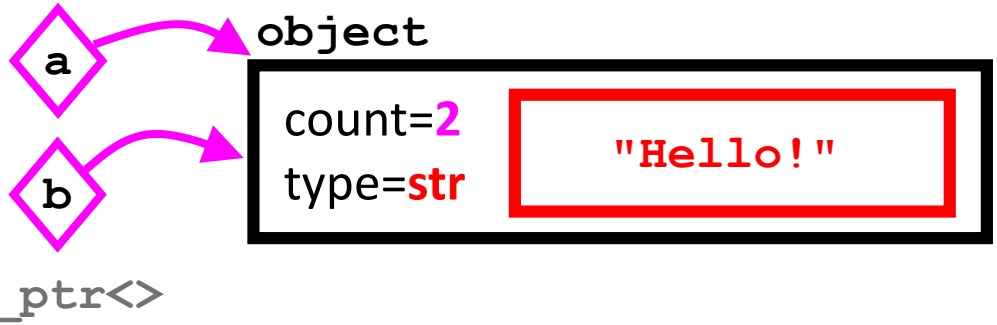

python™
Dynamically typed
Interpreted
CPython (python.org)
Python text
def func(a, b):
c = a + b
print c
return c
CPython compiler
Byte-code
PUSH a
PUSH b
BINARY_ADD
POP c
PUSH c
PRINT_ITEM
PRINT_NEWLINE
PUSH c
RETURN_VALUE
Pops a,b
Adds
Push result
CPython Interpreter

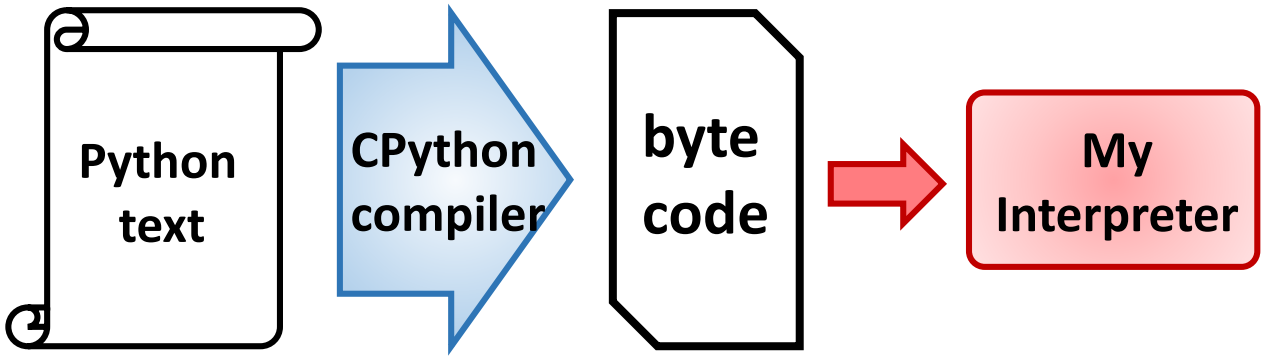
Everything Is an Object



Everything Is Reference Counted

```
a = "Hello!"
b = a
```






```
void test()
{
   Interpreter vm
   Ref module = vm.import(R"**(
      def pyfunc(a,b):
         return a+b
   )**");
   Ref resObj = module.call("pyfunc", 1, 2);
   int result = extract<int>(resObj);
}
```

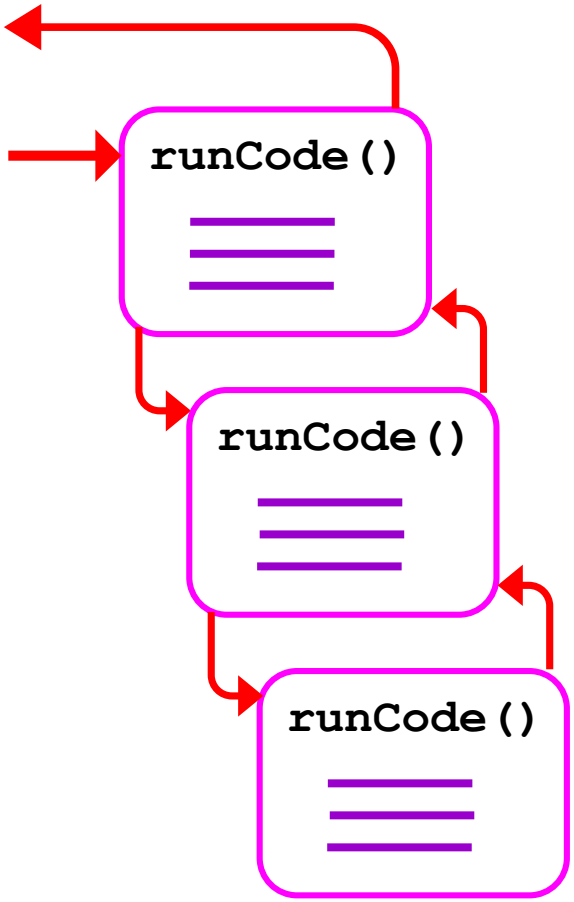
```
int cxxfunc(const string& msg) {
    cout << msg;
    return 42
}

void test()
{
   Interpreter vm
   vm.builtins.def("cxxfunc", cxxfunc);
   Ref module = vm.import(R"**(
       def pyfunc(a,b):
           cxxfunc("Hello!")
           return a+b
   )**");
   Ref resObj = module.call("pyfunc", 1, 2);
}
```

SEAMLESS CONVERSION OF ARGUMENTS !!!

```
struct Thing {
  void action(int a, int b);
};

void test()
{
   Interpreter vm
   Ref m = vm.builtins
   Ref c = m.class_<Thing>("Thing")
               .def("action", &Thing::action);
   Thing t;
   Ref wrap_t = c.wrapInstance(t)
   wrap_t.call("action", 1, 2)
}
```

```
void unpack(vector<Ref>& v) {}

template<typename T, typename ...Args>
void unpack(vector<Ref>& v, const T&& a, const Args&&... args) {
    v.push_back(objectFromT(std::forward<T>(a)));
    unpack(v, args...);
}

Template<typename ...Args>
Ref call(const string& name, const Args&&... args) {
  vector<Ref> argv;
  unpack(argv, args...);
  runCode(lookup(name), argv);
}
```

```
Ref runCode(ByteCode code, vector<Ref> args) {
  Frame f(args)
  int pc = 0
  while(true) {
    byte opcode = code[pc];
    byte param  = code[pc + 1]
    switch(opcode) {
      case PUSH:
        f.stack.push(f.vars[param]);
        break;
      case POP:
        f.vars[param] = f.stack.pop();
        break;
      case JUMP_TO:
        pc = param
        continue;
      case CALL_FUNCTION:
        runCode(pop(), popArgs(param))
        break;
      case RETURN_VALUE:
        return frame.pop();
    }
    pc += 2;
  }
}
```



(boost::intrusive_ptr)

```
struct Object {
    int m_refcount;
};
```

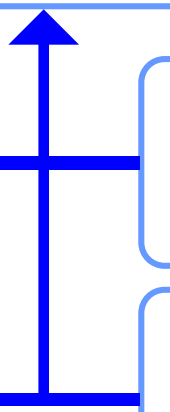
```
struct Ref {
    Object *m_ptr;
};
```

```
struct Str : Object {
    std::string v;
};
```

```
struct List : Object {
    std::vector<Ref> v;
};
```

```
struct Int : Object {
    int v;
};
```

```
struct Dict : Object {
    std::map<int, Ref> v;
};
```



```
template<typename T>
T extract(Ref ref);

template<>
std::string extract(Ref ref) {
    return dynamic_cast<Str>(ref.m_ptr)->v;
}
```

Why?

- Need scripting but don’t want to write a compiler?
- LUA is too strange, Everybody knows python.

CPython	Your Interpreter
~350,000 loc	~2000 loc
Global State	Multiple instance
GIL (global interpreter lock)	Policy for real multi threading
Garbage Collection	- Dump all, start new - Memory snapshot
C API, C code +boost::python	C++11


Gratuitous image of a wild animal