Core C++
2021

Implementing C++ Semantics in Python

Tamir Bahar

© All rights reserved

THE ACADEMIC of TEL AVIV-YAFFO

From C++ Import Magic

Implementing C++ Semantics in Python

Tamir Bahar (He/Him)

@tmr232

Pre-COVID Hairstyle

Before we start, a few questions

- Who uses C++?
- Who uses C++ their main language?
- Who uses Python?
- Who uses Python as their main language?

Your Questions

- Try to write down slide numbers

But Why?

C++

- Low level
- "Expert oriented"
- Slowly becoming "Pythonic"

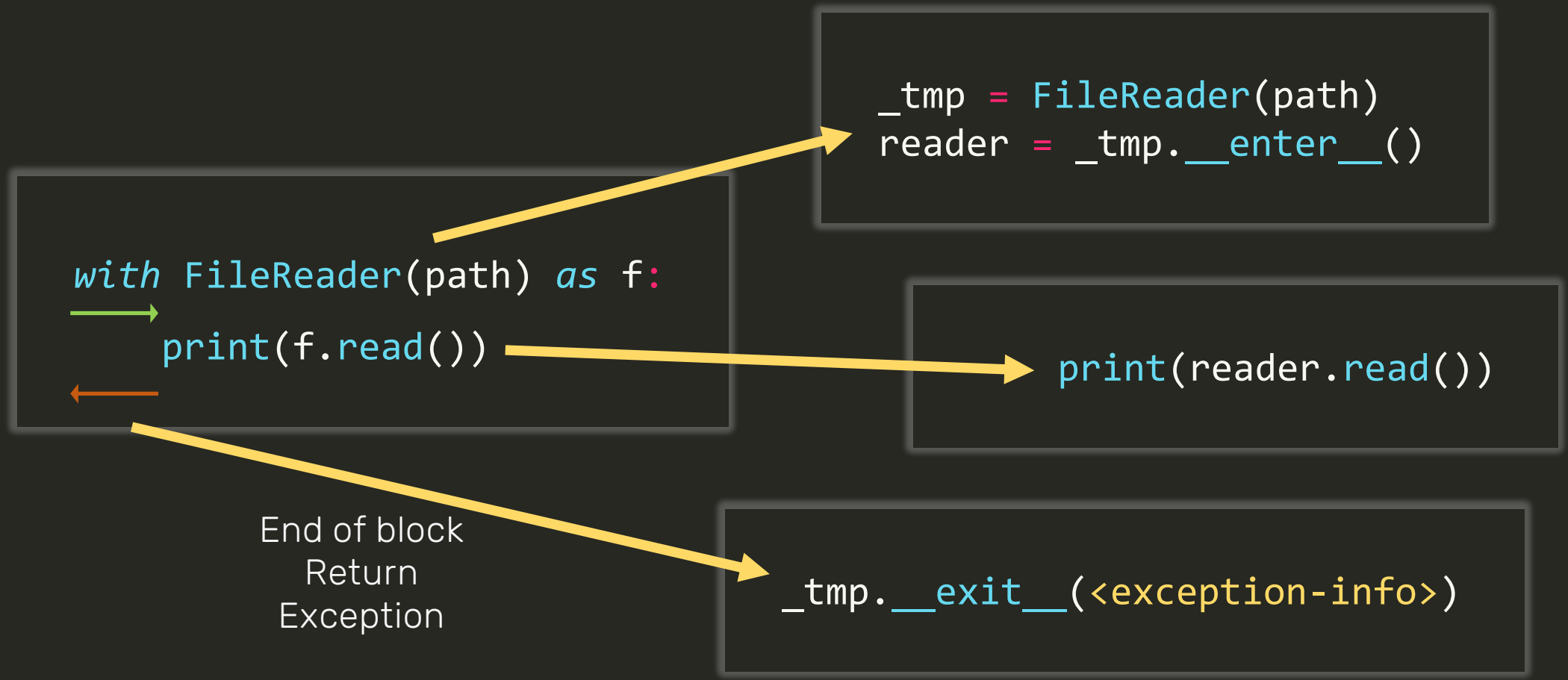
Python

- High Level
- Beginner friendly
- Less footguns

Resource Management

- In C++, all resources are equal
- Python is garbage-collected
- Memory is handled by the language
- Other resources are handled by the programmer
  - Files, sockets, locks, DB connections, etc.

Context Managers



Context Managers, continued

```
class FileReader:
    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.close()

    ...
```

<exception-info>

Real Code - Archive Reader

```
class ArchiveReader:
    def __init__(self, path: str):
        self.data = {}
        with ZipFile(path) as zipfile:
            for name in zipfile.namelist():
                with zipfile.open(name) as f:
                    self.data[name] = f.read()

    def read(self, name):
        return self.data[name]
```

Real Code - Archive Reader

```python
class ArchiveReader:
    def __init__(self, path: str):
        self.data = {}
        with ZipFile(path) as zipfile:
            for name in zipfile.namelist():
                with zipfile.open(name) as f:
                    self.data[name] = f.read()

    def read(self, name):
        return self.data[name]
```

```python
reader = ArchiveReader("corecpp.zip")
print(reader.read("2021"))
```

```
Hello CoreC++!
```

Archive Reader, continued

- Archives got larger
- Time to open archive grows
- Can no longer unzip entire archive in memory
- Need to hold open ZipFile in our Archive Reader

Big Archive Reader

Create

Use

Destruct

```
class BigArchiveReader:
    def __init__(self, path: str):
        self.zipfile = ZipFile(path)

    def read(self, name: str):
        with self.zipfile.open(name) as f:
            return f.read()

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.zipfile.close()
```

Big Archive Reader, continued

- Context managers change interface
- Interface changes propagate
  - Usage
  - Composition

```
reader = ArchiveReader("corecpp.zip")
print(reader.read("2021"))
```

```
with BigArchiveReader("corecpp.zip") as big_reader:
    print(reader.read("2021"))
```

```
...

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.big_reader.close()
...
```

C++ To The Rescue!

Destructors

- C++'s solution to the resource-management problem
- 3 main properties
  - Automatic
  - Composable
  - Implicit

Automatic Invocation

```
{
    auto reader = FileReader(path);

    std::cout << reader.read() << '\n';
}
```

End of block
Return
Exception

```
reader.~FileReader();
```

Seamless Composition

```
class ArchiveReader {
    ...
};
```

```
class BigArchiveReader {
    ZipFile zipfile;
    ...
};
```

```
    auto reader = ArchiveReader(path);
}
```

```
    auto big_reader = BigArchiveReader(path);
}
```

```
~ArchiveReader();
```

```
~BigArchiveReader();
```

```
~ZipFile();
```

Implicit Interfaces

With Destructors

```
{
    auto object = Object();
}
```

Without Destructors

```
{
    auto object = Object();
}
```

Implicit Interfaces

With Destructors

```
{
    auto object = Object();
}
```

Without Destructors

```
{
    auto object = Object();
}
```

- No change in interface or usage
- No change propagation

Our Goal - From This

```
class BigArchiveReader:
    zipfile: ZipFile

    def __init__(self, path: str):
        self.zipfile = ZipFile(path)

    def read(self, name: str):
        with self.zipfile.open(name) as f:
            return f.read()

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.zipfile.close()
```

- 11 lines
- 4 are resource management

Our Goal - To This

```
class BestArchiveReader:
    zipfile: ZipFile

    def BestArchiveReader(self, path: str):
        self.zipfile = ZipFile(path)

    def read(self, name: str):
        with self.zipfile.open(name) as f:
            return f.read()
```

- 7 lines
- 0 are resource management

Our Goal

```
with BigArchiveReader("corecpp.zip") as big_reader:
    print(reader.read("2021"))
```

- From interface pollution
- To normal objects

```
best_reader = BestArchiveReader("corecpp.zip")
print(best_reader.read("2021"))
```

Don't try this at work!

Hacks Ahead!

Greetings!

```
class Greeter:
    def __init__(self, name):
        self.name = name
        print(f"Hello, {self.name}!")

    def __enter__(self):
        return self

    def __exit__(self, e_type, e_val, e_tb):
        print(f"Goodbye, {self.name}.")
```

```
def main():
    with Greeter(1):
        print("We have a greeter!")
```

```
Hello, 1!
We have a greeter!
Goodbye, 1.
```

Automatic

Stacking Dtors

```
def main():
    with Greeter(1):
        print("Hello, Greeters!")
```

```
Hello, 1!
Hello, Greeters!
Goodbye, 1.
```

Stacking Dtors

```
def main():
    with Greeter(1):
        with Greeter(2):
            print("Hello, Greeters!")
```

```
Hello, 1!
Hello, 2!
Hello, Greeters!
Goodbye, 2.
Goodbye, 1.
```

Stacking Dtors

```
def main():
    with Greeter(1):
        with Greeter(2):
            with Greeter(3):
                print("Hello, Greeters!")
```

```
Hello, 1!
Hello, 2!
Hello, 3!
Hello, Greeters!
Goodbye, 3.
Goodbye, 2.
Goodbye, 1.
```

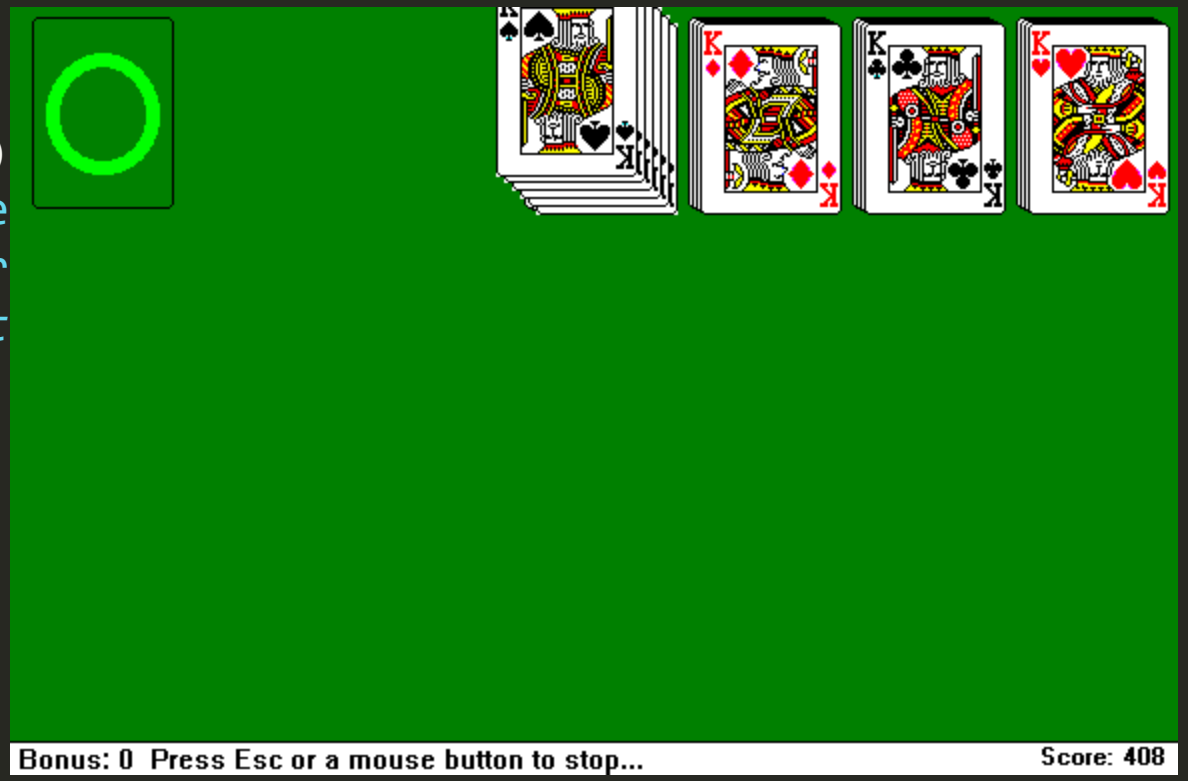
Stacking Dtors

```
def main():
    with Greeter(1):
        with Greeter(2):
            with Greeter(3):
                with Greeter(4):
                    print("Hello, Greeters!")
```

```
Hello, 1!
Hello, 2!
Hello, 3!
Hello, 4!
Hello, Greeters!
Goodbye, 4.
Goodbye, 3.
Goodbye, 2.
Goodbye, 1.
```

Stacking Dtors

```
def main():
    with Greeter(1)
        with Greete
            with Gr
                wit
```



Stacking Dtors

```
def main():
    with Greeter(1):
        with Greeter(2):
            with Greeter(3):
                with Greeter(4):
                    print("Hello, Greeters!")
```

A Proper Stack

```
class DtorScope:
    def __init__(self):
        self.stack = []

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        while self.stack:
            self.stack.pop().__exit__(exc_type, exc_val, exc_tb)

    def push(self, cm):
        self.stack.append(cm)
```

A Proper Stack, continued

```
def main():
    with DtorScope() as dtor_stack:
        greeter1 = Greeter(1)
        dtor_stack.push(greeter1)

        greeter2 = Greeter(2)
        dtor_stack.push(greeter2)
```

```
Hello, 1!
Hello, 2!
Goodbye, 2.
Goodbye, 1.
```

Implicit

~~Removing~~ Hiding Boilerplate

```
def main():
    with DtorScope() as dtor_stack:
        greeter1 = Greeter(1)
        dtor_stack.push(greeter1)

        greeter2 = Greeter(2)
        dtor_stack.push(greeter2)
```

```
def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)
```

A Layer of Indirection

```
class Greeter:
    def __init__(self, name, dtor_stack):
        self.name = name
        print(f"Hello, {self.name}!")

        dtor_stack.push(self)
    ...
```

```
def main():
    with DtorScope() as dtor_stack:
        greeter1 = Greeter(1, dtor_stack)
        greeter2 = Greeter(2, dtor_stack)
```

Another Layer of Indirection

```
def main():
    with DtorScope() as dtor_stack:
        greeter1 = Greeter(1, dtor_stack)
        greeter2 = Greeter(2, dtor_stack)
```

```
def main():
    with DtorScope():
        greeter1 = Greeter(1)
        greeter2 = Greeter(2)
```

Another Layer of Indirection

```
def main():
    with DtorScope() as dtor_stack:
        greeter1 = Greeter(1, dtor_stack)
        greeter2 = Greeter(2, dtor_stack)
```

```
def main():
    with DtorScope():
        greeter1 = Greeter(1)
        greeter2 = Greeter(2)
```

Globals to the Rescue!

```
_dtor_stack = []


def get_dtor_stack():
    return _dtor_stack
```

```
class DtorScope:
    def __init__(self):
        get_dtor_stack().append(self)

    ...

    def __exit__(self, exc_type, exc_val, exc_tb):
        get_dtor_stack().pop()
        ...
    ...
```

```
def push_dtor(cm):
    return get_dtor_stack()[-1].push(cm)
```

Globals to the Rescue!

```
class Greeter:
    def __init__(self, name, dtor_stack):
        dtor_stack.push(self)

        self.name = name
        print(f"Hello, {self.name}!")
    ...
```

Globals to the Rescue!

```
class Greeter:
    def __init__(self, name):
        push_dtor(self)

        self.name = name
        print(f"Hello, {self.name}!")
    ...
```

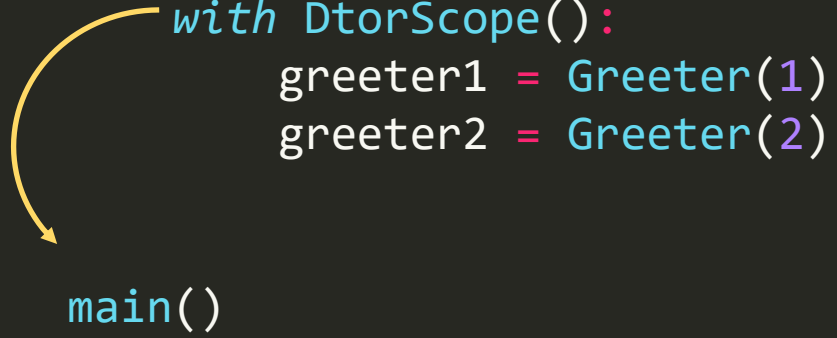
Globals to the Rescue!

```
class Greeter:
    def __init__(self, name):
        push_dtor(self)

        self.name = name
        print(f"Hello, {self.name}!")
    ...
```

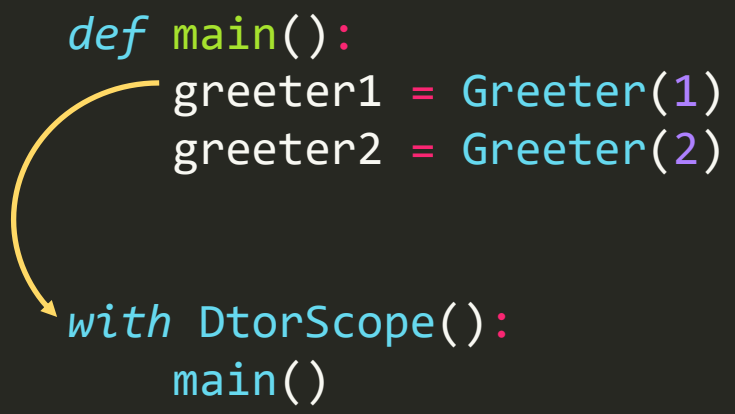
```
def main():
    with DtorScope():
        greeter1 = Greeter(1)
        greeter2 = Greeter(2)
```

```
Hello, 1!
Hello, 2!
Goodbye, 2.
Goodbye, 1.
```

Moving Out


def main():
with DtorScope():
greeter1 = Greeter(1)
greeter2 = Greeter(2)
main()

Moving Out


def main():
greeter1 = Greeter(1)
greeter2 = Greeter(2)
with DtorScope():
main()

Moving Out


```
def call(f, *args, **kwargs):
    with DtorScope():
        return f(*args, **kwargs)


call(main)
```

Moving Out

```
def cpp_function(f):

    def _wrapper(*args, **kwargs):
        with DtorScope():
            return f(*args, **kwargs)

    return _wrapper


scoped_main = cpp_function(main)

scoped_main()
```

Closure

f is captured

Holds the closure

Moving Out

```
def cpp_function(f):

    def _wrapper(*args, **kwargs):
        with DtorScope():
            return f(*args, **kwargs)

    return _wrapper


main = cpp_function(main)

main()
```

Moving Out

Decorator syntax

```
def cpp_function(f):

    def _wrapper(*args, **kwargs):
        with DtorScope():
            return f(*args, **kwargs)

    return _wrapper


main = cpp_function(main)

main()
```

```
@cpp_function
def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)

main()
```

Moving Out

```
@cpp_function
def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)

main()
```

- Declarative
- Clean
- Explicit

Methodic Pause

Import Hacks

Where things get hairy

Basic File Structure

```python
from cpp import cpp_function

from greeter import Greeter


@cpp_function
def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)


main()
```

Import

Usage

Wouldn't it be Nice?

```
from cpp import magic

from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)
```

Modest Beginnings

```
from cpp import magic

from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)


magic()

main()
```

Making Magic

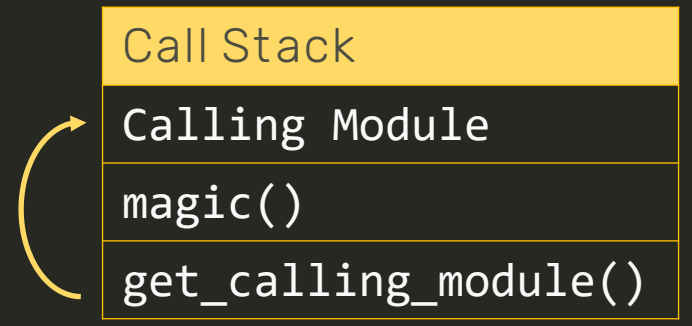
```
def magic():
    calling_module = get_calling_module()
    decorate_module_functions(calling_module)
```

Making Magic

```python
def magic():
    calling_module = get_calling_module()
    decorate_module_functions(calling_module)
```

```python
import inspect

def get_calling_module():
    stack_frame = inspect.stack()[2].frame
    module = inspect.getmodule(stack_frame)
    return module
```

2 places up the callstack



Making Magic

```
def magic():
    calling_module = get_calling_module()
    decorate_module_functions(calling_module)


def decorate_module_functions(module):
    for name, value in inspect.getmembers(module):
        if not inspect.isroutine(value):
            continue

        if inspect.getmodule(value) != module:
            continue

        setattr(module, name, cpp_function(value))
```

All module members

Only functions

Defined in the module

For My Next Trick...

```python
from cpp import magic

from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)


magic()

main()
```

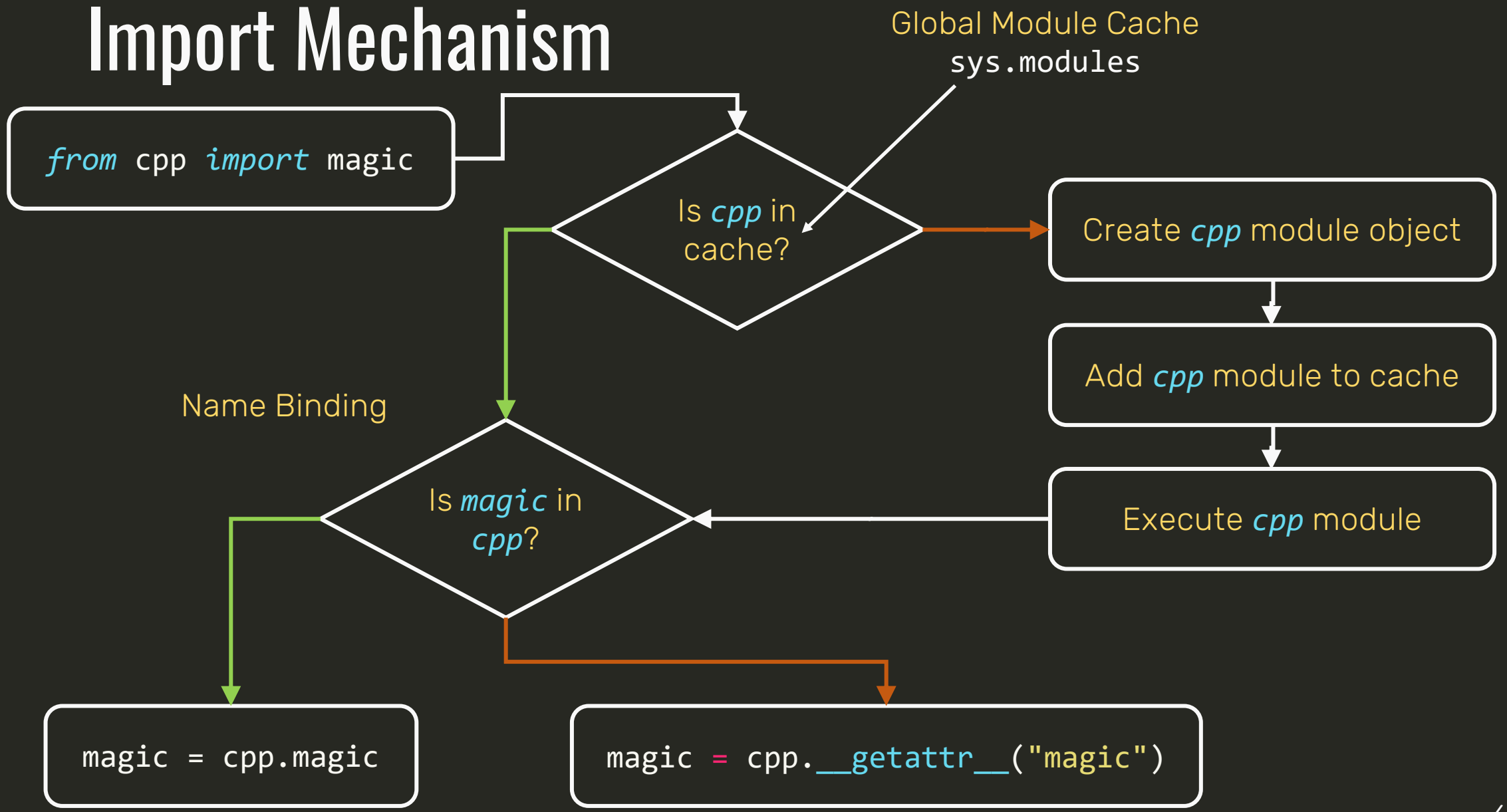
```python
from cpp import magic

from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)



main()
```

For My Next Trick...

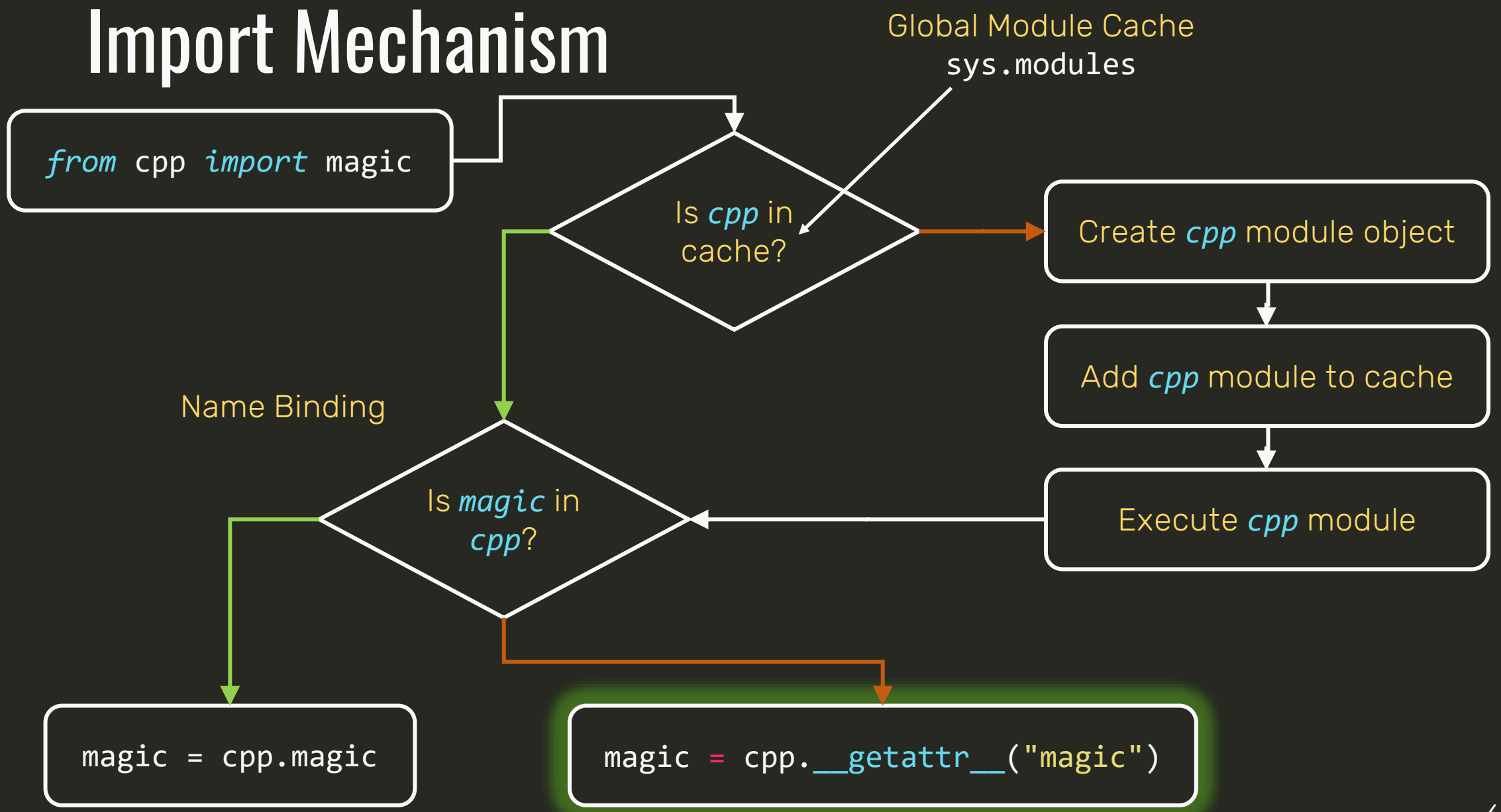
```python
from cpp import magic

from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)


magic()

main()
```

```python
from cpp import magic

from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)



main()
```

Import Mechanism



Import Mechanism



Import *is* a Function Call

```
def _magic():
    calling_module = get_calling_module()
    decorate_module_functions(calling_module)


def __getattr__(name):
    if name != "magic":
        raise AttributeError()

    _magic()
```

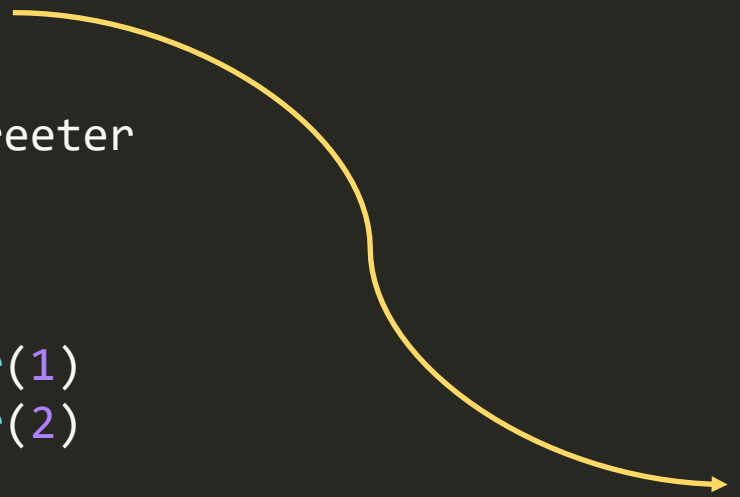
Where is the Magic?

```
from cpp import magic

from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)


main()
```

```
Hello, 1!
Hello, 2!
```

Where is the Magic?

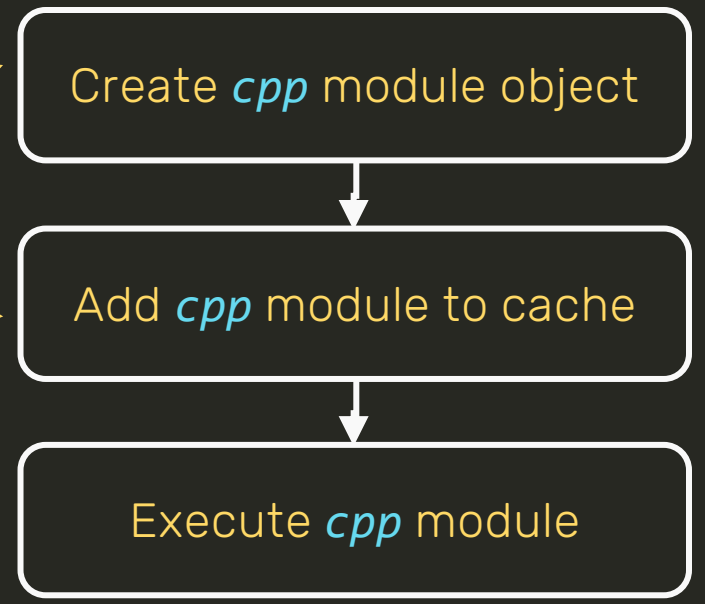
```python
from cpp import magic

from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)


main()
```



```python
from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)


from cpp import magic


main()
```

Parallel Import

```
import importlib.util
import sys

def import_by_path(name: str, path: str):
    spec = importlib.util.spec_from_file_location(name, path)
    module = importlib.util.module_from_spec(spec)
    sys.modules[name] = module
    spec.loader.exec_module(module)
    return module
```



Parallel Import

```
def _magic():
    calling_module = get_calling_module()

    name = calling_module.__name__
    path = calling_module.__file__
    imported_module = import_by_path(name, path)

    decorate_module_functions(imported_module)
```

Parallel Import, continued

```
from cpp import magic

from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)


main()
```

Parallel Import, continued

```python
from cpp import magic

from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)


main()
```

```
RecursionError:
        maximum recursion depth exceeded
        while calling a Python object
```

Break the Loop

```python
IMPORT_FLAG = "__magically_imported__"

def import_by_path(name: str, path: str):
    ...
    sys.modules[name] = module

    setattr(module, IMPORT_FLAG, True)

    spec.loader.exec_module(module)
    return module
```

Set

Check

```python
def _magic():
    ...
    if hasattr(calling_module, IMPORT_FLAG):
        return

    imported_module = import_by_path(name, path)

    decorate_module_functions(imported_module)
```

Break the Loop, continued

```
from cpp import magic

from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)


main()
```

Break the Loop, continued

```
from cpp import magic

from greeter import Greeter


def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)


main()
```

```
IndexError: list index out of range

def push_dtor(cm):
    return get_dtor_stack()[-1].push(cm)
```

Decoration happens here

Main Function

```
def _magic():
    ...

    if imported_module.__name__ == "__main__":
        sys.exit(imported_module.main())
```

Real Magic

```
from cpp import magic

from greeter import Greeter

def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)
```

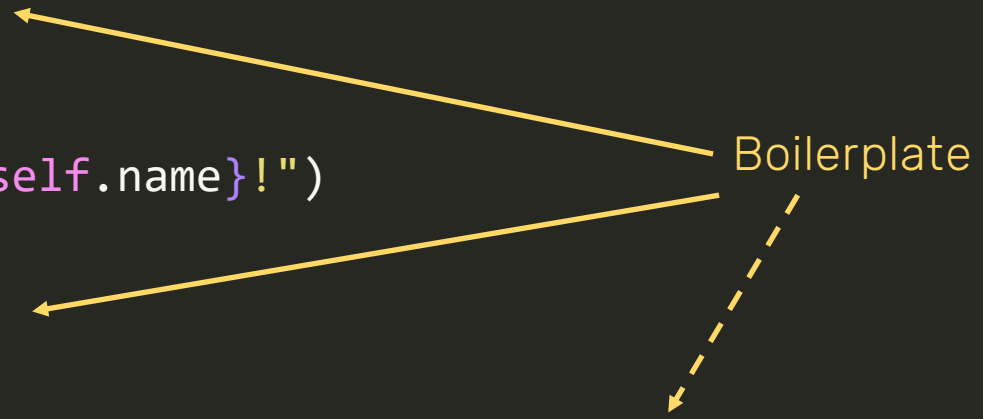
```
Hello, 1!
Hello, 2!
Goodbye, 2.
Goodbye, 1.
```

Methodic Pause

Questions?

Greetings, Again

```
class Greeter:
    def __init__(self, name):
        push_dtor(self)

        self.name = name
        print(f"Hello, {self.name}!")

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(f"Goodbye, {self.name}.")
```



Base Class

```
class Greeter(CppClass):
    def Greeter(self, name):
        self.name = name
        print(f"Hello, {self.name}!")

    def _Greeter(self):
        print(f"Goodbye, {self.name}.")
```

C++ Style Ctor & Dtor

Sorry, no ~

Base Class, implementation

```
class CppClass:
    def __init__(self, *args, **kwargs):
        push_dtor(self)

        ctor = getattr(self, self.__class__.__name__, None)
        if ctor:
            ctor(*args, **kwargs)

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        dtor = getattr(self, "_" + self.__class__.__name__, None)
        if dtor:
            dtor()
```

Only call if exists

Decorated Methods

```
def decorate_object_methods(obj):
    for name, value in inspect.getmembers(obj):
        if name.startswith("__"):
            continue

        if not inspect.isroutine(value):
            continue

        setattr(self, name, cpp_function(value))


class CppClass:
    def __init__(self, *args, **kwargs):
        ...
        decorate_object_methods(self)
```

No special methods

Only functions

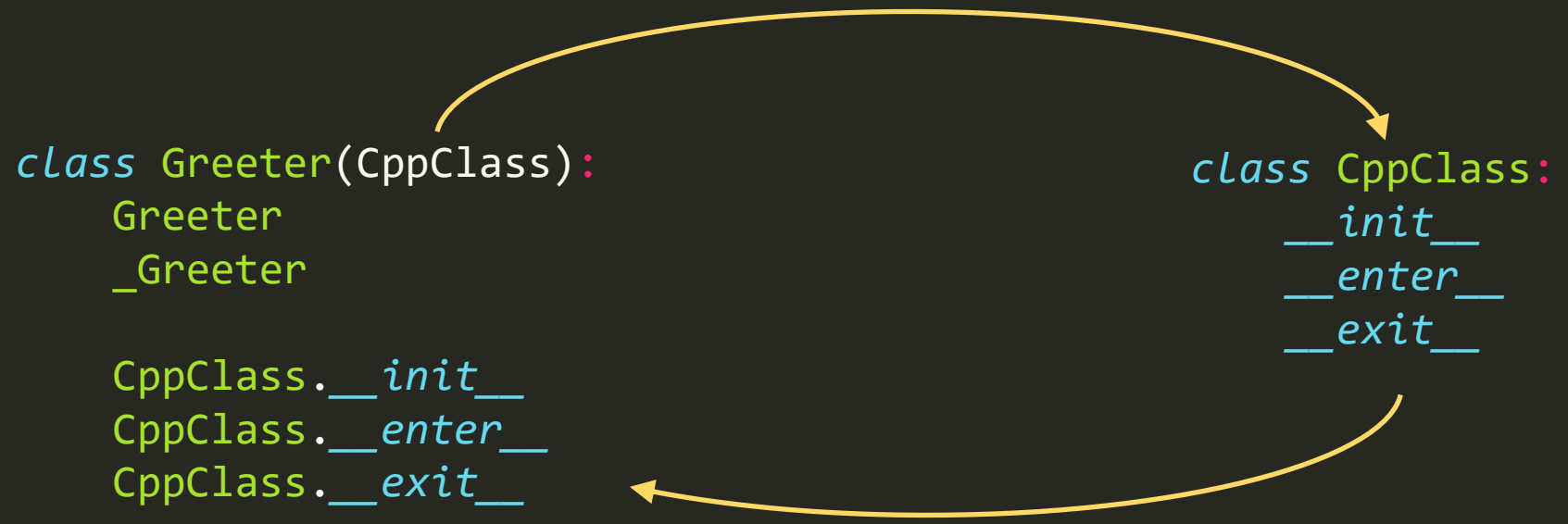
Progress Check

```
class Greeter(CppClass):
    def Greeter(self, name):
        self.name = name
        print(f"Hello, {self.name}!")

    def _Greeter(self):
        print(f"Goodbye, {self.name}.")
```

(More) Problems with Inheritance


```
class Greeter(CppClass):
    def Greeter(self, name):
        self.name = name
        print(f"Hello, {self.name}!")

    def _Greeter(self):
        print(f"Goodbye, {self.name}.")
```

Compositionally Speaking



Compositionally Speaking, continued

```
class Greeter:
    ...


Greeter.__init__ = __init__
Greeter.__enter__ = __enter__
Greeter.__exit__ = __exit__
```

Decorated Classes

```
def cpp_class(cls):
    decorate_object_methods(self)

    def __init__(self, *args, **kwargs):
        ...

    def __enter__(self):
        ...

    def __exit__(self, exc_type, exc_val, exc_tb):
        ...

    cls.__init__ = __init__
    cls.__enter__ = __enter__
    cls.__exit__ = __exit__


return cls
```

```
@cpp_class
class Greeter:
    ...
```

Decorated Classes

```
def cpp_class(cls):
    decorate_object_methods(self)

    def __init__(self, *args, **kwargs):
        ...

    def __enter__(self):
        ...

    def __exit__(self, exc_type, exc_val, exc_tb):
        ...

    cls.__init__ = __init__
    cls.__enter__ = __enter__
    cls.__exit__ = __exit__
    cls.__cpp_class__ = True

return cls
```

```
@cpp_class
class Greeter:
    ...
```

```
def is_cpp_class(obj):
    return hasattr(obj, '__cpp_class__')
```

A little extra

More Magic!

```
def decorate_module_classes(module):
    for name, value in inspect.getmembers(module):
        if not inspect.isclass(value):
            continue

        if inspect.getmodule(value) != module:
            continue

        setattr(module, name, cpp_class(value))


def _magic():
    ...
    decorate_module_classes(imported_module)
    ...
```

Only classes

Defined in the module

Applied Magic

```
from cpp import magic


class Greeter:
    def Greeter(self, name):
        self.name = name
        print(f"Hello, {self.name}!")

    def _Greeter(self):
        print(f"Goodbye, {self.name}.")
```

```
def main():
    greeter1 = Greeter(1)
    greeter2 = Greeter(2)
```

```
Hello, 1!
Hello, 2!
Goodbye, 2.
Goodbye, 1.
```

Methodic Pause

Any Questions?

A Short Recap

- Automatic
  - Dtors are called automatically
- Implicit
  - Just import magic!
  - Functions & classes automatically converted
  - `main()` is automatically called
- Our next stop: Composition

Composition

Looking Back

```
class BigArchiveReader:
    zipfile: ZipFile

    def __init__(self, path: str):
        self.zipfile = ZipFile(path)

    def read(self, name: str):
        with self.zipfile.open(name) as f:
            return f.read()

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.zipfile.close()
```

- 11 lines
- 4 are resource management

Looking Back

```
class BetterArchiveReader:
    zipfile: ZipFile

    def BetterArchiveReader(self, path: str):
        self.zipfile = ZipFile(path)

    def read(self, name: str):
        with self.zipfile.open(name) as f:
            return f.read()

    def _BetterArchiveReader(self):
        self.zipfile.close()
```

- 9 lines
- 2 are resource management

Looking Back, issues

```
class BetterArchiveReader:
    zipfile: ZipFile

    def BetterArchiveReader(self, path: str):
        self.zipfile = ZipFile(path)

    def read(self, name: str):
        with self.zipfile.open(name) as f:
            return f.read()

    def _BetterArchiveReader(self):
        self.zipfile.close()
```

Scope End

Dtor

```
Dtor called twice!

~ZipFile
```

Looking Back, ~~issues~~ solution?

```
class BetterArchiveReader:
    zipfile: ZipFile

    def BetterArchiveReader(self, path: str):
        self.zipfile = ZipFile(path)
        remove_dtor(self.zipfile)

    def read(self, name: str):
        with self.zipfile.open(name) as f:
            return f.read()

    def _BetterArchiveReader(self):
        self.zipfile.close()
```

Remove from Dtor scope

Remove from Dtor Scope

```
class DtorScope:
    stack: list
    ...
    def remove(self, cm):
        self.stack.remove(cm)
```

Equality Based

Remove from Dtor Scope

```
class DtorScope:
    stack: list
    ...
    def remove(self, cm):
        self.stack.remove(
            IdentityComparator(cm)
        )
```

```
class IdentityComparator:
    def __init__(self, obj):
        self.obj = obj

    def __eq__(self, other):
        return self.obj is other
```

operator==

Identity check

Remove from Dtor Scope, continued

```
class BetterArchiveReader:
    zipfile: ZipFile

    def BetterArchiveReader(self, path: str):
        self.zipfile = ZipFile(path)
        remove_dtor(self.zipfile)

    def read(self, name: str):
        with self.zipfile.open(name) as f:
            return f.read()

    def _BetterArchiveReader(self):
        self.zipfile.close()
```

A Case for Getters & Setters

```
def get_zipfile(self):
    return getattr(self, "zipfile")


def set_zipfile(self, zipfile):
    old = getattr(self, "zipfile", None)

    if is_cpp_class(old):
        old.__exit__(None, None, None)

    if is_cpp_class(zipfile):
        remove_dtor(zipfile)

    setattr(self, "zipfile", zipfile)
```

Just return

Destruct old value

Handle new value

Descriptors

```
BetterArchiveReader.zipfile.__set_name__(BetterArchiveReader, "zipfile")
```

```
class BetterArchiveReader:
    zipfile = CppMember()

    def BetterArchiveReader(self, path):
        self.zipfile = ZipFile(path)

    def read(self, name: str):
        with self.zipfile.open(name) as f:
            ...

    def _BetterArchiveReader(self):
        self.zipfile.close()
```

```
zipfile.__set__(self, ZipFile(path))
```

```
zipfile.__get__(self)
```

Descriptors, continued

```
class CppMember:
    def __set_name__(self, owner, name):
        self.private_name = "_" + name

    def __get__(self, instance, owner=None):
        return getattr(instance, self.private_name)

    def __set__(self, instance, value):
        old = getattr(instance, self.private_name, None)

        ...

        setattr(instance, self.private_name, value)
```

Save & prefix member name

Use member name

Remove the Dtor

```
class BetterArchiveReader:
    zipfile = CppMember()

    def BetterArchiveReader(self, path):
        self.zipfile = ZipFile(path)

    def read(self, name: str):
            ...

    def _BetterArchiveReader(self):
        self.zipfile.close()
```

Should be implicit

Remove the Dtor, continued

```
def __exit__(self, exc_type, exc_val, exc_tb):
    ...

    for name, value in reversed(inspect.getmembers(self)):
        if name.startswith("_"):
            continue

        if not is_cpp_class(value):
            continue

        value.__exit__(None, None, None)
```

Reverse order

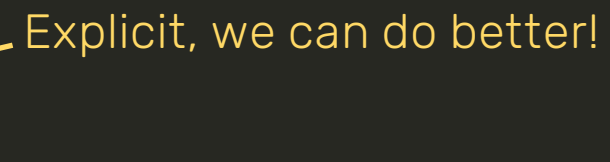
Avoid prefixed members

Has Dtor?

Final Touches

```
class BetterArchiveReader:
    zipfile = CppMember()

    def BetterArchiveReader(self, path):
        self.zipfile = ZipFile(path)

    def read(self, name: str):
            ...
```

Final Touches




```
class BetterArchiveReader:
    zipfile = CppMember()

    def BetterArchiveReader(self, path):
        self.zipfile = ZipFile(path)

    def read(self, name: str):
            ...
```

Type Annotations

```
class BetterArchiveReader:
    zipfile: ZipFile
```

- Do nothing
- Stored in `__annotations__`

Type Annotations, continued

```
def create_members(cls):
    member_names = list(getattr(cls, "__annotations__", {}))

    for name in member_names:
        member = CppMember()
        member.__set_name__(cls, name)
        setattr(cls, name, member)

    setattr(cls, "__member_names__", member_names)


def cpp_class(cls):
    ...
    create_members(cls)
    ...
```

Must call manually

Save for later

Type Annotations, continued

```
def __exit__(self, exc_type, exc_val, exc_tb):
    ...

    for name in reversed(self.__member_names__):
        value = getattr(self, name, None)

        if is_cpp_class(value):
            value.__exit__(None, None, None)
```

Traverse only members

Finally - Best Archive Reader

```
class BestArchiveReader:
    zipfile: ZipFile

    def BestArchiveReader(self, path: str):
        self.zipfile = ZipFile(path)

    def read(self, name: str):
        with self.zipfile.open(name) as f:
            return f.read()
```

- 7 lines
- 0 are resource management

Wrap Up

- Automatic
  - Dtors are called when/where needed
- Composable
  - Members don't add boilerplate
- Implicit*
  - No extra code
  - No change in interfaces
  - No interface pollution!

* Assuming the entire project uses cpp...

Questions?

Thanks

- Barak Itkin
- Adi Shavit
- Inbal Levi

Extras

Extras

- Return
- This
- Member Access Specifiers


Adi Shavit
@AdiShavit
Replying to @tmr232
Plan:
1. Submit a talk to @corecpp
2. Get accepted
3. Deliver the talk with the man-bun.
4. Poll the audience.
GIF
11:46 AM · Jun 8, 2021
4
Copy link to Tweet
Tweet your reply