DEVELOPMENT & DEPLOYMENT ENVIRONMENT (PART 2)

C++ AND PYTHON

Calle Rosenquist, Nov 2019

ME

- Joined ESO in 2015
- Background in IT security, distributed systems, vision systems
- ELT
  - Build System
  - Observation Coordination Framework
  - Application Framework
  - RTC Toolkit

OUTLINE

- C++ 17 & Python 3.5
  - Notable Features
  - Tips/Best Practices
- Bridging C++ and Python

C++ 17 FEATURES

- std::optional
- std::variant
- std::error_code
- noexcept
- [[nodiscard]]
- <random>
- <filesystem>
- <regexp>

C++ 17 NOTABLE FEATURES

- Move Semantics
- Lambda Expressions
- `std::chrono`

MOVE SEMANTICS

- Transfer of ownership
- Enables strong ownership types
- Faster code

```
class vector { T* data; size_t size; ...};

std::vector<BigObj> a;
std::vector<BigObj> b = a; // copies all elements
std::vector<BigObj> c = std::move(b); // copies data pointer, siz

std::unique_ptr<BigObj> ptr1 = std::make_unique<BigObj>();
std::unique_ptr<BigObj> ptr2 = std::move(ptr1); // transfers owne
```

LAMBDA EXPRESSIONS

- Anonymous functions
- Can capture variables
- Convertible to `std::function`

```
auto closure = [](std::string_view msg) {
    std::cout << "Hello " << msg;
};
closure("there"); // prints "Hello there"

auto closure_with_capture = [closure]() {
    closure("world");
};
closure_with_capture(); // prints "Hello world"

std::function<void()> func = closure_with_capture;
func(); // prints "Hello world"
```

LAMBDA EXPRESSIONS

```
boost::future<int> future = some_operation();
future.then([](boost::future<int> fut){
    ...
});

void upper(std::string_view str) {
    std::tranform(std::begin(str), std::end(str), std::begin(str)
                  [](unsigned char c) -> unsigned char {
                      return std::toupper(c);
                  });
}
```

std::chrono

Type safe date/time operations

- Timepoint
- Duration
- Units
- Conversion

```
std::this_thread::sleep_for(2s); // 2s == std::chrono::seconds(2)

auto now = std::chrono::high_resolution_clock::now();
auto future = now + 100ms; // 100ms == std::chrono::milliseconds(

std::this_thread::sleep_until(future);
```

C++ BEST PRACTICES AND TIPS

PREFER `std::array` OVER C ARRAYS

```
constexpr auto NUM = 10u;

void foo(int a[NUM]);
int a[NUM-5];
foo(a); // Not an error

void foo(std::array<int, NUM>& a); // do
std::array<int, NUM-5> a;
foo(a); // error

struct Foo {
    std::array<int, NUM> a; // do
    int a[NUM]; // don't
};
```

override

```
class Foo {
    virtual void setUp();
};

class Bar : public Foo {
    void setup() override; // error
}
```

`std::unique_ptr` OR `std::shared_ptr`

When in doubt, use `std::unique_ptr`

```
std::unique_ptr<Foo> exclusive = std::make_unique<Foo>();
// transfer ownerhip to shared_ptr
std::shared_ptr<Foo> shared = std::move(exclusive);
```

CONSTANTS

Use `constexpr` and digit separators:

```
namespace foo {
    constexpr double PI = 3.14...;
    constexpr std::string_view STR = "foo";

    constexpr uint64_t TWO_M = 2'000'000;
    constexpr uint64_t MY_BITS = 0x0000'dead'beef'0000;
    constexpr std::byte MY_BYTE = 0b0000'1111u;
}
```

PYTHON

PYTHON

NOTABLE FEATURES

- Asyncio
- Type hints
- Keyword only arguments

asyncio

Asynchronous I/O with coroutines

```
import asyncio
async def say(*, msg, delay):
    await asyncio.sleep(delay)
    print(msg)

event_loop.run_until_complete(
    say('hello world', 2))
```

TYPE HINTS

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```

KEYWORD ONLY ARGUMENTS

```
def sort(lst, *, reverse=None):
    ...

sort([2,1,3]) # Ok
sort([2,1,3], reverse=True) # Ok
sort([2,1,3], True) # Error, not a kwarg


def only(*, foo, bar):
    """This function only accepts keyword args"""
    ...
```

PYTHON

BEST PRACTICES AND TIPS

CONTEXT MANAGERS

Prevents resource leaks.

```
def func(path: str) -> None:
    handle = open(path)
    this_function_might_raise(handle) # Oops
    handle.close()

def func(path: str) -> None:
    with open(path) as handle:
        this_function_might_raise(handle)
```

DEFAULT ARGUMENTS

```
# Don't
def foo(arg=Bar()):
    print(id(arg))
    arg.on = True # Oops, might modify default argument
foo() # prints xxxx42
foo() # prints xxxx42

def foo(arg=None):
    if arg is None:
        arg = Bar()
    ...
```

EXTENDING PYTHON WITH C++ AND PYBIND11

```
#include <pybind11/pybind11.h>

int add(int i, int j) {
    return i + j;
}

PYBIND11_MODULE(example, m) {
    m.doc() = "pybind11 example plugin"; // optional module docst
    m.def("add", &add, "A function which adds two numbers");
}
```

```
# example/wscript
from wtools.modules import declare_cshlib
declare_cshlib(target = 'example',
               features = 'pyext',
               use = 'pybind11',
               install_path = '${PYTHONDIR}')
```

EXTENDING PYTHON WITH C++ AND PYBIND11

```
$ python
>>> import example
>>> example.add(1, 2)
3
>>>
```

QUESTIONS?

- https://isocpp.github.io/CppCoreGuidelines

QUESTIONS?

- https://isocpp.github.io/CppCoreGuidelines