MIXING C++ & PYTHON II: PYBIND11

IGOR SADCHENKO

IGOR.SADCHENKO@GMAIL.COM

WARGAMING.NET

ABOUT PYTHON

PYTHON

WARGAMING.NET
LET'S BATTLE

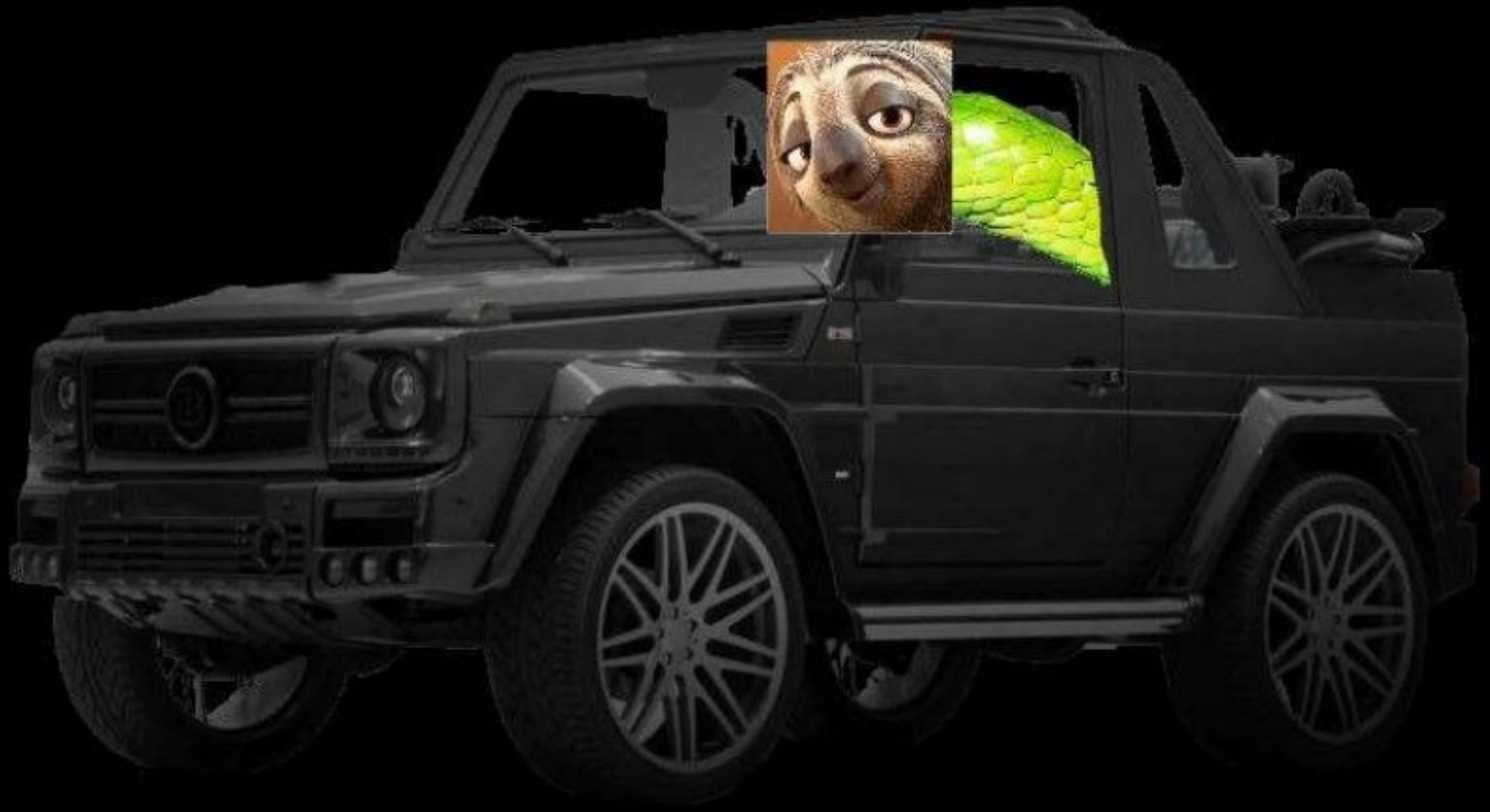
- **Python** is a high-level general purpose programming language that focuses on developer productivity improving and code readability.
- The syntax of the core Python is minimalistic. At the same time, the standard library includes a large amount of useful functions.
- Python supports multiple programming paradigms, including structured, object-oriented, functional, imperative, and aspect-oriented.
- The main architectural features – dynamic typing, automatic memory management, full introspection, handling mechanism exception, multi-threaded computing and comfortable high-level data structure support.
- The disadvantages include low performance and the lack of real multi-threading (GIL)

C++ & PYTHON

WARGAMING.NET
LET'S BATTLE

- Python and C++ easily complement each other.

- Python gives you rapid development and flexibility
- C++ gives you speed and industrial strength tools.



INTRO

PYBIND11

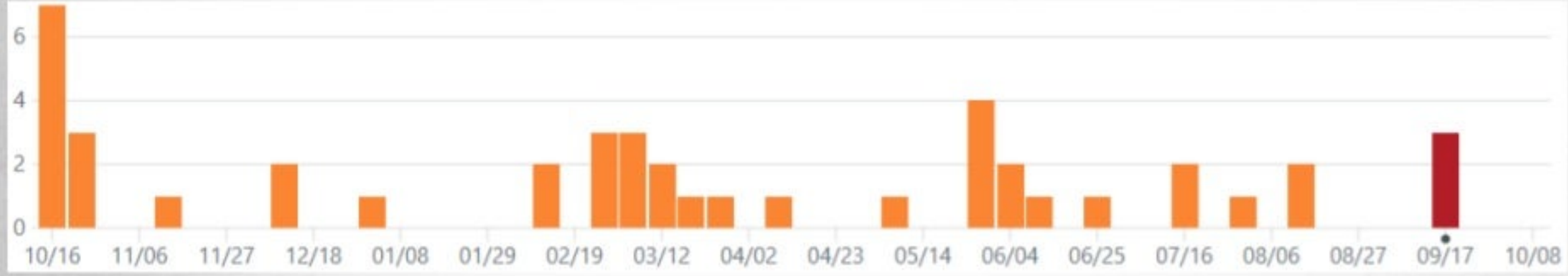
WARGAMING.NET
LET'S BATTLE

- **pybind11** is a lightweight header-only library that exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code.
- Think of this library as a tiny self-contained version of Boost.Python with everything stripped away that isn't relevant for binding generation.
- Without comments, the core header files only require ~4K lines of code and depend on Python (2.7 or 3.x, or PyPy2.7 >= 5.7) and the C++ standard library. This compact implementation was possible thanks to some of the new C++11 language features (specifically: tuples, lambda functions and variadic templates).
- Since its creation, this library has grown beyond Boost.Python in many ways, leading to dramatically simpler binding code in many common situations.

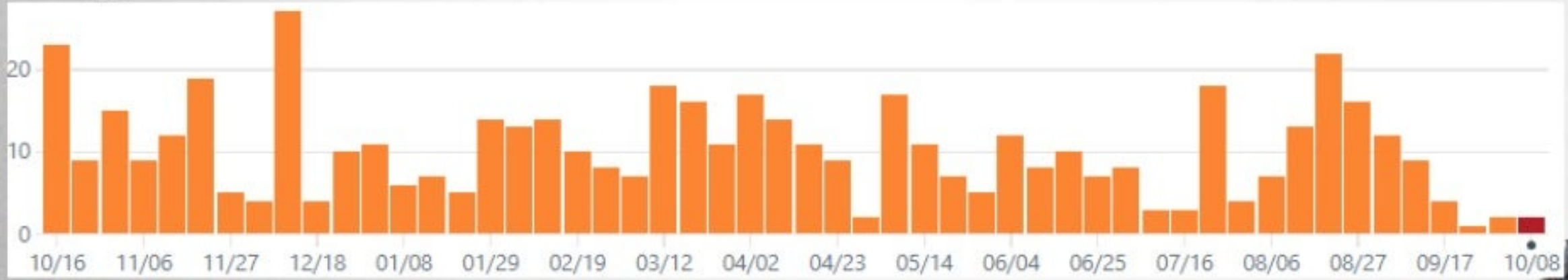
PYBIND11

WARGAMING.NET
LET'S BATTLE

› Boost.Python



› pybind11



ADVANTAGES

WARGAMING.NET
LET'S BATTLE

- Python 2.7, 3.x, and PyPy (PyPy2.7 >= 5.7) are supported with an implementation-agnostic interface.
- It is possible to bind C++11 lambda functions with captured variables.
- Using move constructors/assignment whenever possible to efficiently transfer custom data types.
- It's easy to expose the internal storage of custom data types through Pythons' buffer protocols.
- Automatic vectorization of functions for NumPy array arguments.
- Python's slice-based access and assignment operations can be supported with just a few lines of code.

ADVANTAGES

WARGAMING.NET
LET'S BATTLE

- Everything is contained in just a few header files.
- Binaries and compile time are generally smaller by factor 2 vs. to equivalent bindings of Boost.Python.
- Function signatures are precomputed at compile time (using constexpr), leading to smaller binaries.
- With little extra effort, C++ types can be pickled and unpickled similar to regular Python objects.
- Simple package installation: ***pip install pybind11***

SUPPORTED COMPILERS

WARGAMING.NET
LET'S BATTLE

- Clang/LLVM 3.3 or newer (for Apple clang 5.0.0 or newer)
- GCC 4.8 or newer
- Microsoft Visual Studio 2015 Update 3 or newer
- Intel C++ compiler 16 or newer (15 with a workaround)
- Cygwin/GCC (tested on 2.5.1)

CORE FEATURES

FEATURES

WARGAMING.NET
LET'S BATTLE

- Functions accepting and returning custom data structures per value, reference, or pointer
- Instance methods and static methods
- Overloaded functions
- Instance attributes and static attributes
- Arbitrary exception types
- Enumerations
- Callbacks

FEATURES

WARGAMING.NET
LET'S BATTLE

- Iterators and ranges
- Custom operators
- Single and multiple inheritance
- STL data structures
- Iterators and ranges
- Smart pointers with reference counting like std::shared_ptr
- Internal references with correct reference counting
- C++ classes with virtual (and pure virtual) methods can be extended in Python

DETAILS

SIMPLE EXAMPLE

WARGAMING.NET
LET'S BATTLE

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

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

PYBIND11_MODULE(pybind_integration, m)
{
    m.doc() = "pybind11 example plugin"; // optional docstring
    m.def("add", &add, "Adds two numbers");
}
```

SIMPLE EXAMPLE

WARGAMING.NET
LET'S BATTLE

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




PYBIND11_MODULE(pybind_integration, m)
{
    m.doc() = "pybind11 example plugin"; // optional docstring
    m.def("add", [] (int i, int j) {return i + j;},"Adds two numbers“);
}
```

C++11

SIMPLE EXAMPLE

WARGAMING.NET
LET'S BATTLE

```
>>> import pybind_integration

>>> add = pybind_integration.add
>>> add(2, 5)
7L
```

```
>>> help(add)
Help on built - in function add in module pybind_integration:
add(...)
    add(arg0: int, arg1: int) -> int
    Adds two numbers

>>> add(4.5, 5.5)
Traceback(most recent call last):
    ...
    ...
    add(4.5, 5.5)
TypeError: add(): incompatible function arguments. The following argument types are supported:
    1.(arg0: int, arg1: int) -> int
Invoked with: 4.5, 5.5
```

SIMPLE EXAMPLE

```
>>> import pybind_integration

>>> add = pybind_integration.add
>>> add(2, 5)
7L
```

```
>>> help(add)
Help on built - in function add in module pybind_integration:
add(...)
    add(arg0: int, arg1: int) -> int
    Adds two numbers
```

```
>>> add(4.5, 5.5)
Traceback(most recent call last):
    ...
    ...
    add(4.5, 5.5)
TypeError: add(): incompatible function arguments. The following argument types are supported:
    1.(arg0: int, arg1: int) -> int
Invoked with: 4.5, 5.5
```

SIMPLE EXAMPLE

WARGAMING.NET
LET'S BATTLE

```
>>> import pybind_integration

>>> add = pybind_integration.add
>>> add(2, 5)
7L

>>> help(add)
Help on built - in function add in module pybind_integration:
add(...)
    add(arg0: int, arg1: int) -> int
    Adds two numbers
```

```
>>> add(4.5, 5.5)
Traceback(most recent call last):
    ...
    ...
    add(4.5, 5.5)
TypeError: add(): incompatible function arguments. The following argument types
are supported:
    1.(arg0: int, arg1: int) -> int
Invoked with: 4.5, 5.5
```

BUILDING A MODULE

WARGAMING.NET
LET'S BATTLE

- Linux

```
$ c++ -O3 -Wall -shared -std=c++11 -fPIC `python3 -m pybind11 --includes`
pybind_integration.cpp -o pybind_integration`python3-config --extension-suffix`
```

- Mac OS

```
$ c++ -O3 -Wall -shared -std=c++11 -undefined dynamic_lookup `python3 -m pybind11 --
includes` pybind_integration.cpp -o pybind_integration`python3-config --extension-suffix`
```

- Windows

BUILDING A MODULE

WARGAMING.NET LET'S BATTLE

- Linux

```
$ c++ -O3 -Wall -shared -std=c++11 -fPIC `python3 -m pybind11 --includes`  pybind_integration.cpp -o pybind_integration`python3-config --extension-suffix`
```

- Mac OS

```
$ c++ -O3 -Wall -shared -std=c++11 -undefined dynamic_lookup `python3 -m pybind11 --includes` pybind_integration.cpp -o pybind_integration`python3-config --extension-suffix`
```

- ~~Windows~~ Cmake

```
cmake_minimum_required(VERSION 2.8.12)

set(PYBIND11_CPP_STANDARD /std:c++11)
set(PYBIND11_PYTHON_VERSION=3.6)

project(pybind_integration)

add_subdirectory(pybind11)
pybind11_add_module(pybind_integration MODULE pybind_integration.cpp)
```

BUILDING A MODULE

WARGAMING.NET
LET'S BATTLE

- Building with setuptools
  - https://github.com/pybind/python_example
- Building with cppimport
  - https://github.com/tbenthompson/cppimport
- Generating binding code automatically(LLVM/Clang)
  - http://cppbinder.readthedocs.io/en/latest/about.html

CLASS EXAMPLE

WARGAMING.NET
LET'S BATTLE

```
#include <string>

#define INITIAL_LIVES 3u

struct Tank {
    std::string name;
    unsigned lives;
    int x;
    int y;
};
```

CLASS EXAMPLE

pybind11

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

PYBIND11_MODULE(tank, m) {
    pybind11::class_<Tank>(m, "Tank")
        .def_readonly("name", &Tank::name)
        .def_readonly("lives", &Tank::lives)
        .def_readonly("x", &Tank::x)
        .def_readonly("y", &Tank::y);
}
```

CLASS EXAMPLE

pybind11

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

PYBIND11_MODULE(tank, m) {
    pybind11::class_<Tank>(m, "Tank")
        .def_readonly("name", &Tank::name)
        .def_readwrite("lives", &Tank::lives)
        .def_readwrite("x", &Tank::x)
        .def_readwrite("y", &Tank::y);
}
```

CLASS EXAMPLE

WARGAMING.NET
LET'S BATTLE

```
#include <string>

#define INITIAL_LIVES 3u

struct Tank {
    std::string name;
    unsigned lives;
    int x;
    int y;
    Tank(std::string name_, unsigned lives_ = INITIAL_LIVES, int x_ = 0, int y_ = 0)
        : name{ name_ }, lives{ lives_ }, x{ x_ }, y{ y_ }
    {}
    Tank() : Tank("IS") {}
};
```

CLASS EXAMPLE

pybind11

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

PYBIND11_MODULE(tank, m) {
    pybind11::class_<Tank>(m, "Tank")
        .def(pybind11::init<>())
        .def(pybind11::init<std::string>())
        .def(pybind11::init<std::string, unsigned>())
        .def(pybind11::init<std::string, unsigned, int>())
        .def(pybind11::init<std::string, unsigned, int, int>())
        .def_readonly("name", &Tank::name)
        .def_readonly("lives", &Tank::lives)
        .def_readonly("x", &Tank::x)
        .def_readonly("y", &Tank::y);
}
```

CLASS EXAMPLE

WARGAMING.NET
LET'S BATTLE

```
#include <string>

#define INITIAL_LIVES 3u

struct Tank {
    std::string name;
    unsigned lives;
    int x;
    int y;
    Tank(std::string name_, unsigned lives_ = INITIAL_LIVES, int x_ = 0, int y_ = 0)
        : name{ name_ }, lives{ lives_ }, x{ x_ }, y{ y_ }
    {}
    Tank() : Tank("IS")
    {}
    bool is_dead() const { return lives == 0u; }
};
```

CLASS EXAMPLE

pybind11

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

PYBIND11_MODULE(tank, m) {
      pybind11::class_<Tank>(m, "Tank")
            .def(pybind11::init<>())
            .def(pybind11::init<std::string>())
            .def(pybind11::init<std::string, unsigned>())
            .def(pybind11::init<std::string, unsigned, int>())
            .def(pybind11::init<std::string, unsigned, int, int>())
            .def_readonly("name", &Tank::name)
            .def_readonly("lives", &Tank::lives)
            .def_readonly("x", &Tank::x)
            .def_readonly("y", &Tank::y)
            .def_property_readonly("is_dead", &Tank::is_dead)
}
```

CLASS EXAMPLE

WARGAMING.NET
LET'S BATTLE

```
#include <string>

#define INITIAL_LIVES 3u

struct Tank {
        std::string name;
        unsigned lives;
        int x;
        int y;
        Tank(std::string name_, unsigned lives_ = INITIAL_LIVES, int x_ = 0, int y_ = 0)
                : name{ name_ }, lives{ lives_ }, x{ x_ }, y{ y_ }
        {}
        Tank() : Tank("IS")
        {}
        bool is_dead() const { return lives == 0u; }
        std::string to_string() const {
                return "<" + name + ":" + std::to_string(lives) +
                        " [" + std::to_string(x) + ", " + std::to_string(y) + "]>";
        }
};
```

CLASS EXAMPLE

pybind11

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

PYBIND11_MODULE(tank, m) {
    pybind11::class_<Tank>(m, "Tank")
        .def(pybind11::init<>())
        .def(pybind11::init<std::string>())
        .def(pybind11::init<std::string, unsigned>())
        .def(pybind11::init<std::string, unsigned, int>())
        .def(pybind11::init<std::string, unsigned, int, int>())
        .def_readonly("name", &Tank::name)
        .def_readonly("lives", &Tank::lives)
        .def_readonly("x", &Tank::x)
        .def_readonly("y", &Tank::y)
        .def_property_readonly("is_dead", &Tank::is_dead)
        .def("__repr__", &Tank::to_string);
}
```

CLASS EXAMPLE

WARGAMING.NET
LET'S BATTLE

```
#include <string>

#define INITIAL_LIVES 3u

struct Tank {
        std::string name;
        unsigned lives;
        int x;
        int y;
        Tank(std::string name_, unsigned lives_ = INITIAL_LIVES, int x_ = 0, int y_ = 0)
                : name{ name_ }, lives{ lives_ }, x{ x_ }, y{ y_ }
        {}
        Tank() : Tank("IS")
        {}
        bool is_dead() const { return lives == 0u; }
        std::string to_string() const {
                return "<" + name + ":" + std::to_string(lives) + " [" + std::to_string(x) + ", " + std::to_string(y) + "]>";
        }
};

bool operator==(const Tank& t1, const Tank& t2) {
    return t1.name == t2.name;
}
```

CLASS EXAMPLE

pybind11

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

PYBIND11_MODULE(tank, m) {
    pybind11::class_<Tank>(m, "Tank")
        .def(pybind11::init<>())
        .def(pybind11::init<std::string>())
        .def(pybind11::init<std::string, unsigned>())
        .def(pybind11::init<std::string, unsigned, int>())
        .def(pybind11::init<std::string, unsigned, int, int>())
        .def_readonly("name", &Tank::name)
        .def_readonly("lives", &Tank::lives)
        .def_readonly("x", &Tank::x)
        .def_readonly("y", &Tank::y)
        .def_property_readonly("is_dead", &Tank::is_dead)
        .def("__eq__", [](const Tank& l, const Tank& r) { return l == r; }, pybind11::is_operator())
        .def("__repr__", &Tank::to_string);
}
```

CLASS EXAMPLE

pybind11

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

#include <pybind11/operators.h>

PYBIND11_MODULE(tank, m) {
    pybind11::class_<Tank>(m, "Tank")
        .def(pybind11::init<>())
        .def(pybind11::init<std::string>())
        .def(pybind11::init<std::string, unsigned>())
        .def(pybind11::init<std::string, unsigned, int>())
        .def(pybind11::init<std::string, unsigned, int, int>())
        .def_readonly("name", &Tank::name)
        .def_readonly("lives", &Tank::lives)
        .def_readonly("x", &Tank::x)
        .def_readonly("y", &Tank::y)
        .def_property_readonly("is_dead", &Tank::is_dead)
        .def(pybind11::self == pybind11::self)
        .def("__repr__", &Tank::to_string);
}
```

CLASS EXAMPLE

pybind11

WARGAMING.NET LET'S BATTLE

```
PYBIND11_MODULE(tank, m) {
pybind11::class_<Tank>(m, "Tank")
...
    .def(pybind11::pickle(
        [](const Tank& t) { /* __getstate__ */
            return pybind11::make_tuple(t.name, t.lives, t.x, t.y);
        },
        [](pybind11::tuple t) {/* __setstate__ */
            if (t.size() != 4) throw std::runtime_error("Invalid data!");
            Tank tank(t[0].cast<std::string>(), t[1].cast<unsigned>(), t[2].cast<int>(), t[3].cast<int>());
            return tank;
        }
    ));
}
```

CLASS EXAMPLE

python

```
>>> from tank import Tank

>>> Tank()
<IS:3 [0, 0]>

>>> is2 = Tank("IS-2")
>>> is2.name
'IS-2'

>>> is2.is_dead
False

>>> is2 == Tank()
False
```

FUNCTIONS

WARGAMING.NET LET'S BATTLE

- Docstrings can be set by passing string literals to def().
- Arguments can be named via py::arg("...").

```
m.def("shoot", [](const std::string& name) {
        pybind11::print("Didn't penetrate the armor of " + name + ".");
    },
    "Shoot a tank.", pybind11::arg("name")
);
```

FUNCTIONS

WARGAMING.NET LET'S BATTLE

- Docstrings can be set by passing string literals to def().
- Arguments can be named via py::arg("...").

```
m.def("shoot", [](const std::string& name) {
                pybind11::print("Didn't penetrate the armor of " + name + ".");
      },
      "Shoot a tank.", pybind11::arg("name")
);
```

```
>>> shoot('IS')
Didn't penetrate the armor of IS.
>>> help(shoot)
Help on built-in function shoot in module tank:
shoot(...)
    shoot(name: unicode) -> None
    Shoot a tank.
```

FUNCTIONS

WARGAMING.NET LET'S BATTLE

› Default arguments.

```
m.def("shoot", [](const std::string& name, unsigned times) {
        if (times > 3)
            pybind11::print("Kill " + name + ".")
        else
            pybind11::print("Didn't penetrate " + name + ".");
    },
    "Shoot a tank.", pybind11::arg("name"), pybind11::arg("times") = 1
);
```

FUNCTIONS

WARGAMING.NET LET'S BATTLE

› Default arguments.

```
m.def("shoot", [](const std::string& name, unsigned times) {
            if (times > 3)
                  pybind11::print("Kill " + name + ".")
            else
                  pybind11::print("Didn't penetrate " + name + ".");
    },
    "Shoot a tank.", pybind11::arg("name"), pybind11::arg("times") = 1
);
```

```
>>> shoot("IS")
Didn't penetrate IS.
>>> shoot("IS-2", 4)
Kill IS-2.
```

FUNCTIONS

WARGAMING.NET LET'S BATTLE

› Variadic positional and keyword arguments.

```
m.def("count_args", [](pybind11::args a, pybind11::kwargs kw) {
    pybind11::print(a.size(), "args,", kw.size(), "kwargs");
});
```

```
>>> count_args(14, 10, 2017, corehard='autumn')
3 args, 1 kwargs
```

FUNCTIONS

WARGAMING.NET LET'S BATTLE

› Python objects as arguments.

```
m.def("count_tanks", [](pybind11::list list) {
    int n = 0;
    for (auto item : list)
        if (pybind11::isinstance<Tank>(item))
            ++n;
    return n;
});
```

```
>>> from tank import count_tanks

>>> count_tanks([Tank("IS-2"), Tank(), 1, "IS-7"])
2
```

FUNCTIONS

WARGAMING.NET LET'S BATTLE

- Function overloading .

```
m.def("go_to", [](int x, int y) { return "go_to int"; });
m.def("go_to", [](double x, double y) { return "go_to double"; });
```

```
>>> go_to(25, 4)
'go_to int'
>>> go_to(3.14, 3.14)
'go_to double'
```

FUNCTIONS

WARGAMING.NET
LET'S BATTLE

- Callbacks.

```
int func_arg(const std::function<int(int)> &f) {
    return f(10);
}
```

FUNCTIONS

WARGAMING.NET LET'S BATTLE

› Callbacks.

```
int func_arg(const std::function<int(int)> &f) {
        return f(10);
}
```

```
std::function<int(int)> func_ret(const std::function<int(int)> &f) {
    return [f](int i) {return f(i) + 1;};
}
```

FUNCTIONS

WARGAMING.NET LET'S BATTLE

› Callbacks.

```
int func_arg(const std::function<int(int)> &f) {
        return f(10);
}
std::function<int(int)> func_ret(const std::function<int(int)> &f) {
        return [f](int i) {return f(i) + 1;};
}
```

```
pybind11::cpp_function func_cpp() {
    return pybind11::cpp_function(
        [](int i) { return i + 1; }, pybind11::arg("number")
    );
}
```

FUNCTIONS

WARGAMING.NET
LET'S BATTLE

› Callbacks.

```
int func_arg(const std::function<int(int)> &f) {
        return f(10);
}
std::function<int(int)> func_ret(const std::function<int(int)> &f) {
        return [f](int i) {return f(i) + 1;};
}
pybind11::cpp_function func_cpp() {
        return pybind11::cpp_function(
                [](int i) { return i + 1; }, pybind11::arg("number")
        );
}
```

```
#include <pybind11/functional.h>
PYBIND11_MODULE(example, m) {
    m.def("func_arg", &func_arg);
    m.def("func_ret", &func_ret);
    m.def("func_cpp", &func_cpp);
}
```

FUNCTIONS

WARGAMING.NET LET'S BATTLE

- Callbacks.

```
>>> import example
>>> def square(i):
...     return i * i
...
>>> example.func_arg(square)
100L
>>> square_plus_1 = example.func_ret(square)
>>> square_plus_1(4)
17L
>>> plus_1 = func_cpp()
>>> plus_1(number=43)
44L
```

ENUMS

WARGAMING.NET LET'S BATTLE

```
struct Tank {
    ...
    enum Type { HEAVY, LIGHT };
};
```

```
pybind11::class_<Tank> cls(m, "Tank");
pybind11::enum_<Tank::Type>(cls, "Type")
    .value("HEAVY", Tank::Type::HEAVY)
    .value("LIGHT", Tank::Type::LIGHT)
    .export_values();
```

› Notes: pybind11 enums are not ints

EMBEDDING THE INTERPRETER

WARGAMING.NET LET'S BATTLE

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

int main() {
    pybind11::scoped_interpreter guard{}; // start the interpreter and keep it alive

    pybind11::exec(R"(
        kwargs = dict(name="IS", number=667)
        message = "Shoot {name} with id {number}".format(**kwargs)
        print(message)
    )");
}
```

EMBEDDING THE INTERPRETER

WARGAMING.NET
LET'S BATTLE

```
cmake_minimum_required(VERSION 3.0)
project(embed_itnterpreter)

find_package(pybind11 REQUIRED)  # or `add_subdirectory(pybind11)`

add_executable(embed_itnterpreter main.cpp)
target_link_libraries(embed_itnterpreter PRIVATE pybind11::embed)
```

EMBEDDING THE INTERPRETER

WARGAMING.NET LET'S BATTLE

- Importing modules
  - `pybind11::module sys = pybind11::module::import("sys");`
  - `pybind11::print(sys.attr("path"));`
- Adding embedded modules
  -
```
PYBIND11_EMBEDDED_MODULE(fast_calc, m) {
    // 'm' is a 'py::module' which is used to bind functions and classes
    m.def("add", [](int i, int j) {
        return i + j;
    });
}
```
-
```
auto fast_calc = pybind11::module::import("fast_calc");
auto result = fast_calc.attr("add")(1, 2).cast<int>();
```

SUMMARY

CONCLUSION

WARGAMING.NET
LET'S BATTLE

- pybind11
- simple to use
- efficient
- supported by the community

LINKS

WARGAMING.NET
LET'S BATTLE

- python
  - https://www.python.org
  - https://docs.python.org/3/extending/index.html
  - https://docs.python.org/3/c-api/index.html
- pybind11
  - https://github.com/pybind/pybind11
  - http://pybind11.readthedocs.io/en/stable

THANK YOU FOR YOUR ATTENTION!

WARGAMING.NET

LET'S BATTLE

ANY QUESTIONS?

IGOR SADCHENKO

software developer

igor.sadchenko@gmail.com

+375 33 642 92 91

https://www.facebook.com/WargamingMinsk

https://www.linkedin.com/company/wargaming-net

wargaming.com