Integrando Python e C++ com pybind11

Tarcísio Fischer

Agenda

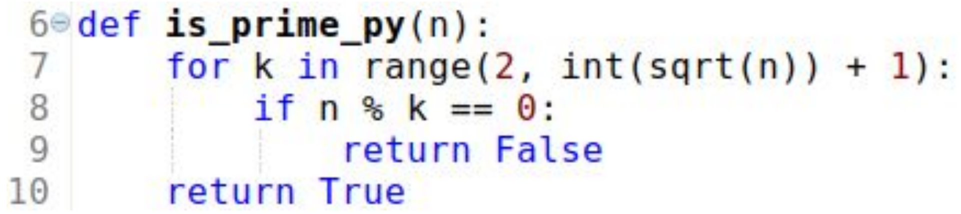
- Motivação
- Introdução
- Básicos
- Features
- Exemplo

Motivação

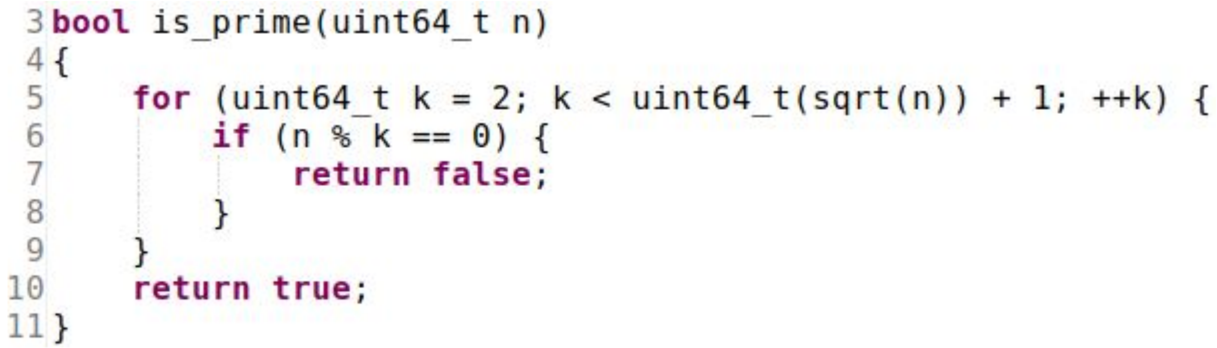
- Performance (Python pode ser lento e Numpy as vezes não é o bastante)

Motivação

- Performance (Python pode ser lento e Numpy as vezes não é o bastante)

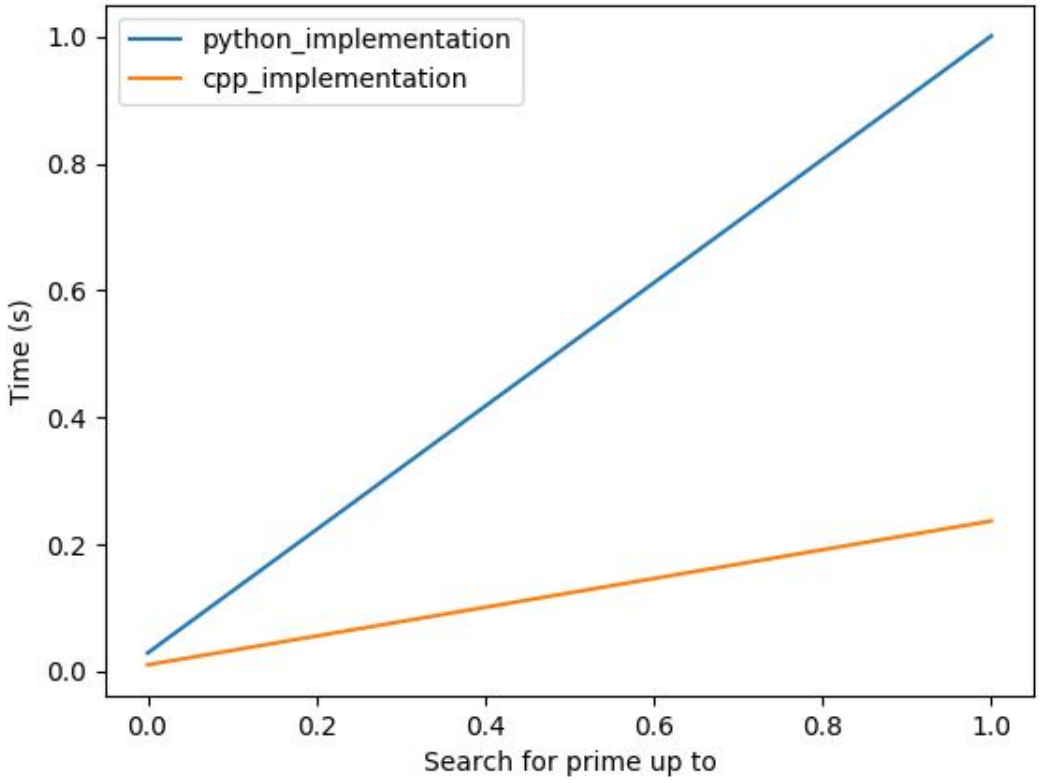


```python
def is_prime_py(n):
    for k in range(2, int(sqrt(n)) + 1):
        if n % k == 0:
            return False
    return True
```



```cpp
bool is_prime(uint64_t n)
{
    for (uint64_t k = 2; k < uint64_t(sqrt(n)) + 1; ++k) {
        if (n % k == 0) {
            return false;
        }
    }
    return true;
}
```

Comparação: Tempo de execução testando números primos até certo valor



Motivação

- Performance (Python é lento e Numpy as vezes não é o bastante)
- Aproveitar libs C++

Motivação

- Performance (Python é lento e Numpy as vezes não é o bastante)
- Aproveitar libs C++

Motivação

- Performance (Python é lento e Numpy as vezes não é o bastante)
- Aproveitar libs C++
- Trazer código legado C++ para Python

Introdução - O que é a Pybind11

- Biblioteca C++ que permite expor código C++ em Python e vice versa

Introdução - O que é a Pybind11

- Biblioteca C++ que permite expor código C++ em Python e vice versa
- Alternativas?
    - Boost::Python - Tempo de compilação lento, não está mais sendo mantida

Introdução - O que é a Pybind11

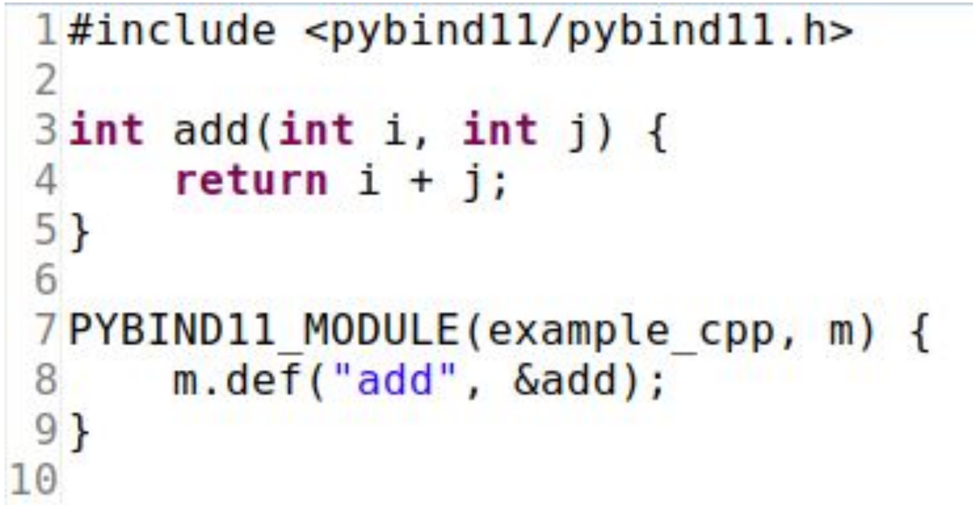
- Biblioteca C++ que permite expor código C++ em Python e vice versa
- Alternativas?
    - Boost::Python - Tempo de compilação lento, não está mais sendo mantida
    - Cython - Difícil manter; Não tem boas ferramentas (IDE/Debugger)

Introdução - O que é a Pybind11

- Biblioteca C++ que permite expor código C++ em Python e vice versa
- Alternativas?
    - Boost::Python - Tempo de compilação lento, não está mais sendo mantida
    - Cython - Difícil manter; Não tem boas ferramentas (IDE/Debugger)
    - Python C API - Muito Low-Level

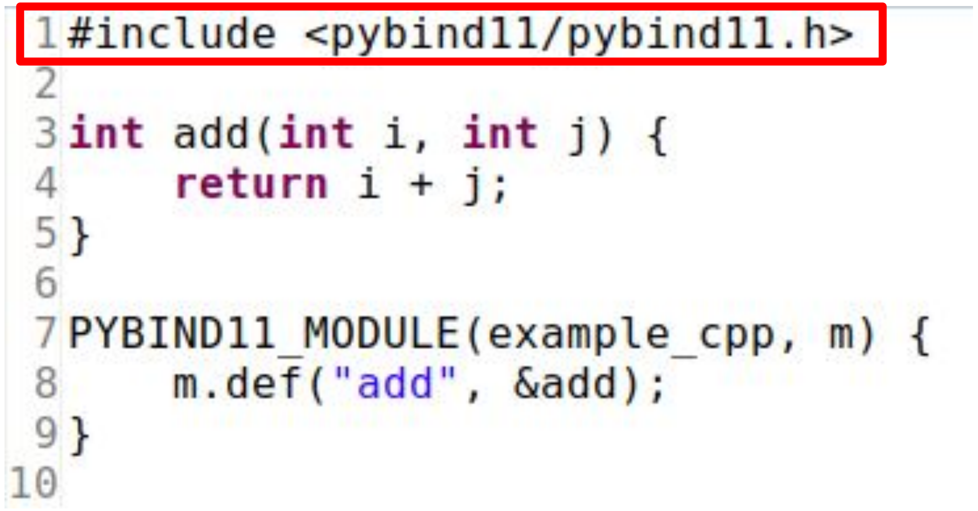
SHOW ME MORE
SHOW ME MORE
makeameme.org

Básicos - Funções

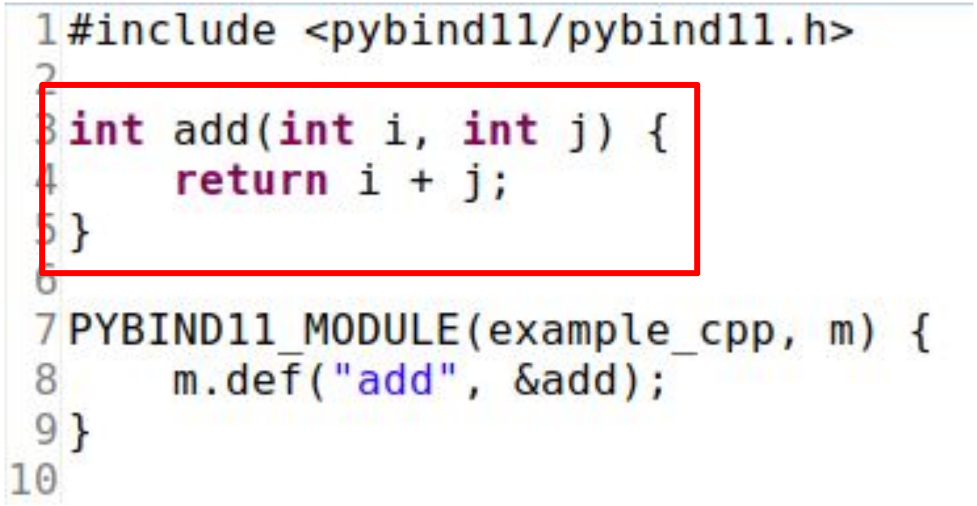


```cpp
#include <pybind11/pybind11.h>

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

PYBIND11_MODULE(example_cpp, m) {
    m.def("add", &add);
}

```

Básicos - Funções

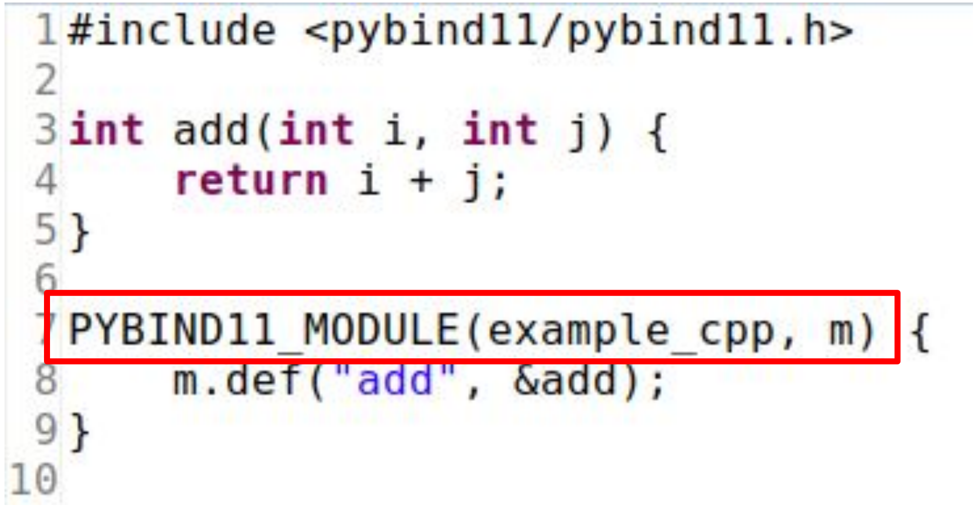

```cpp
#include <pybind11/pybind11.h>

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

PYBIND11_MODULE(example_cpp, m) {
    m.def("add", &add);
}

```

Básicos - Funções



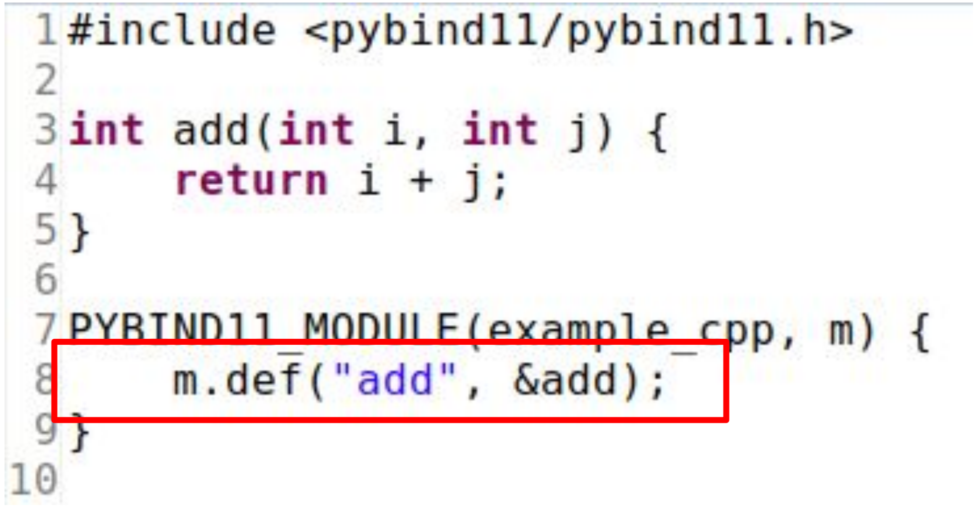
```cpp
#include <pybind11/pybind11.h>

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

PYBIND11_MODULE(example_cpp, m) {
    m.def("add", &add);
}

```

Básicos - Funções



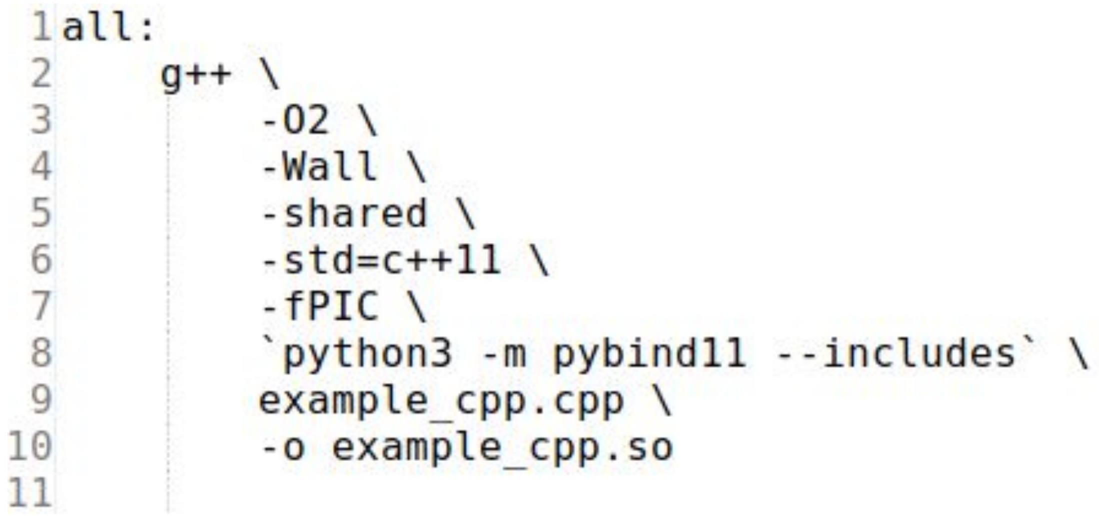
```cpp
#include <pybind11/pybind11.h>

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

PYBIND11_MODULE(example_cpp, m) {
    m.def("add", &add);
}

```

Básicos - Funções



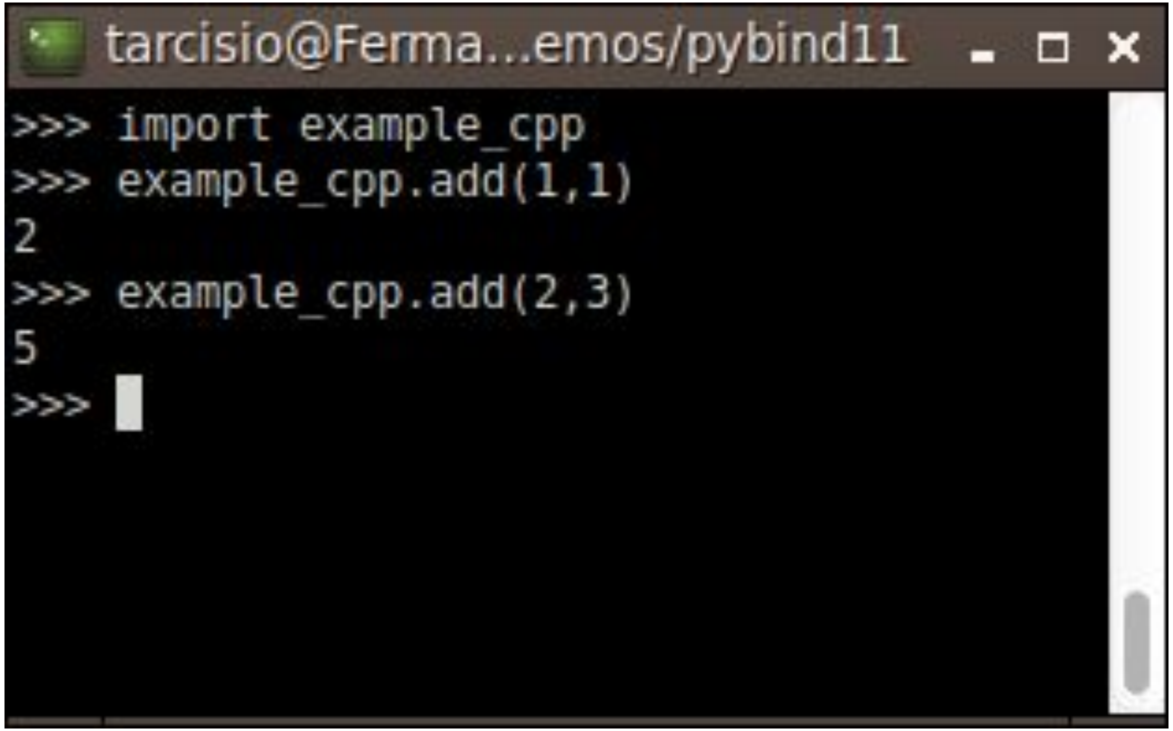
```cpp
#include <pybind11/pybind11.h>

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

PYBIND11_MODULE(example_cpp, m) {
    m.def("add", &add);
}

```

Básicos - Funções - Possível makefile (básico)*



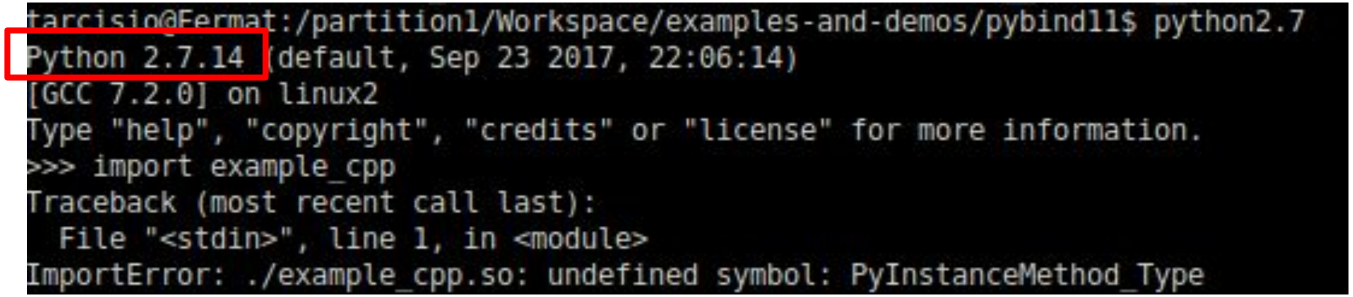
```
all:
	g++ \
		-O2 \
		-Wall \
		-shared \
		-std=c++11 \
		-fPIC \
		`python3 -m pybind11 --includes` \
		example_cpp.cpp \
		-o example_cpp.so

```

Básicos - Funções - Uso em python

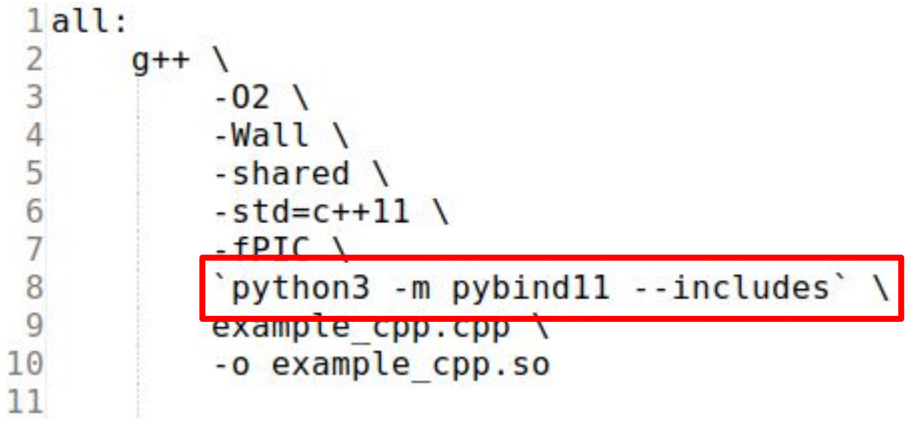


```
>>> import example_cpp
>>> example_cpp.add(1,1)
2
>>> example_cpp.add(2,3)
5
>>> 
```

Básicos - Funções - Import error

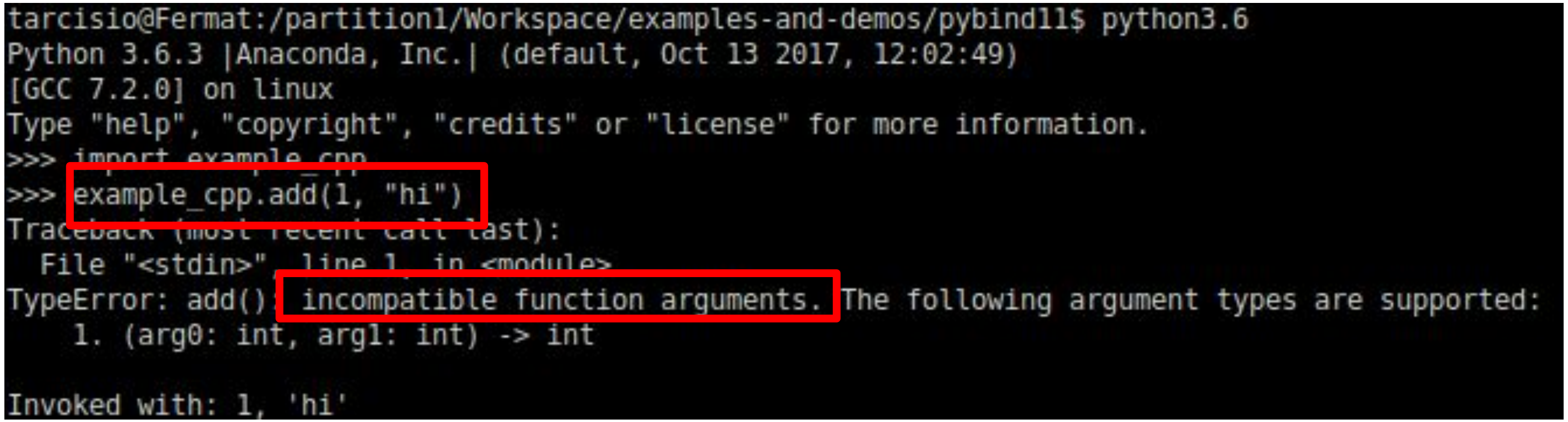


```
tarcisio@Fermat:/partition1/Workspace/examples-and-demos/pybind11$ python2.7
Python 2.7.14 (default, Sep 23 2017, 22:06:14)
[GCC 7.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import example_cpp
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: ./example_cpp.so: undefined symbol: PyInstanceMethod_Type
```



```
all:
	g++ \
		-O2 \
		-Wall \
		-shared \
		-std=c++11 \
		-fPIC \
		`python3 -m pybind11 --includes` \
		example_cpp.cpp \
		-o example_cpp.so

```

Básicos - Funções - Tipos



```
tarcisio@Fermat:/partition1/Workspace/examples-and-demos/pybind11$ python3.6
Python 3.6.3 |Anaconda, Inc.| (default, Oct 13 2017, 12:02:49)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import example_cpp
>>> example_cpp.add(1, "hi")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: add(): incompatible function arguments. The following argument types are supported:
    1. (arg0: int, arg1: int) -> int

Invoked with: 1, 'hi'
```

WE ARE NOT
IMPRESSED
memegenerator.net

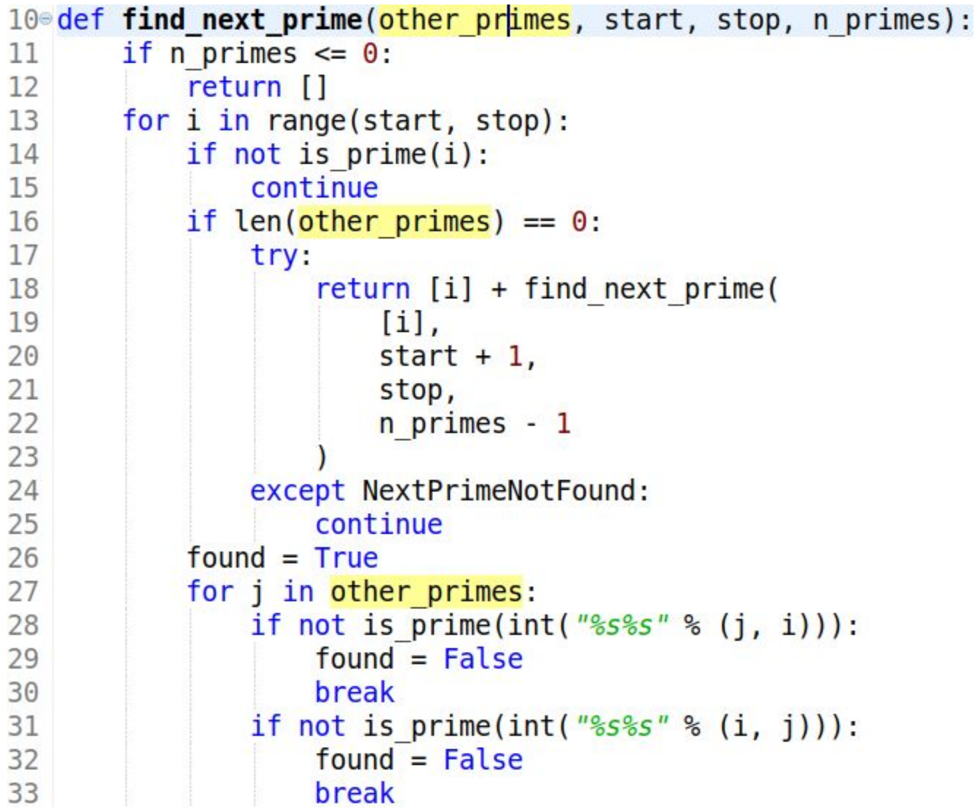
Exemplo 1 (Project Euler, problem 60):

Encontre **5** primos cuja concatenação entre eles sempre gera novos primos.

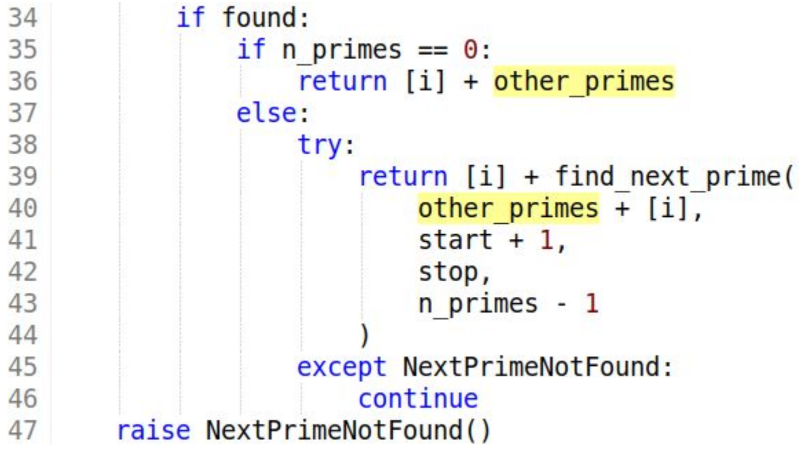
Exemplo (Para **4** primos):
[3, 7, 109, e 673] são primos
37 e 73 são primos
3109 e 1093 são primos
7109 e 1097 são primos (etc…)

https://projecteuler.net/problem=60

Exemplo 1:

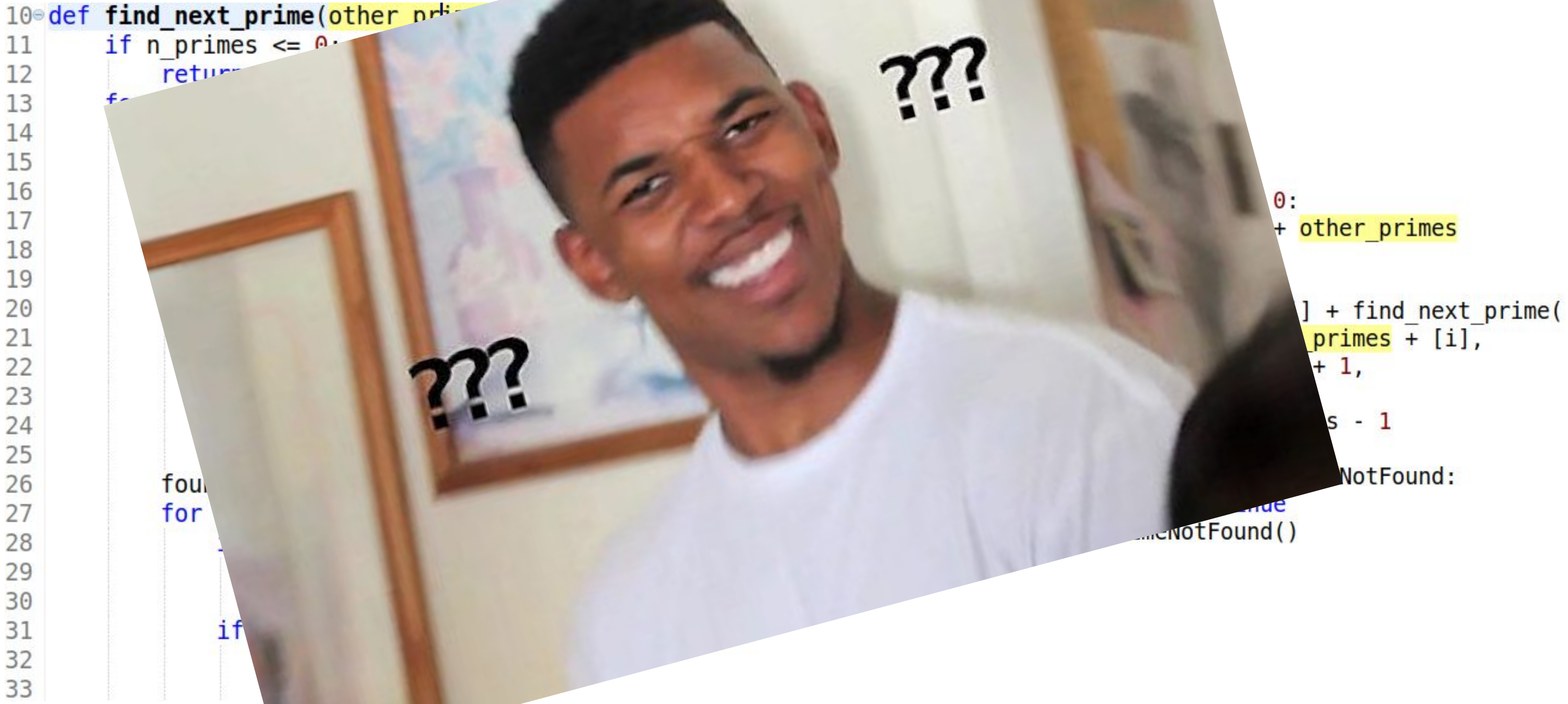


```python
def find_next_prime(other_primes, start, stop, n_primes):
    if n_primes <= 0:
        return []
    for i in range(start, stop):
        if not is_prime(i):
            continue
        if len(other_primes) == 0:
            try:
                return [i] + find_next_prime(
                    [i],
                    start + 1,
                    stop,
                    n_primes - 1
                )
            except NextPrimeNotFound:
                continue
        found = True
        for j in other_primes:
            if not is_prime(int("%s%s" % (j, i))):
                found = False
                break
            if not is_prime(int("%s%s" % (i, j))):
                found = False
                break
```

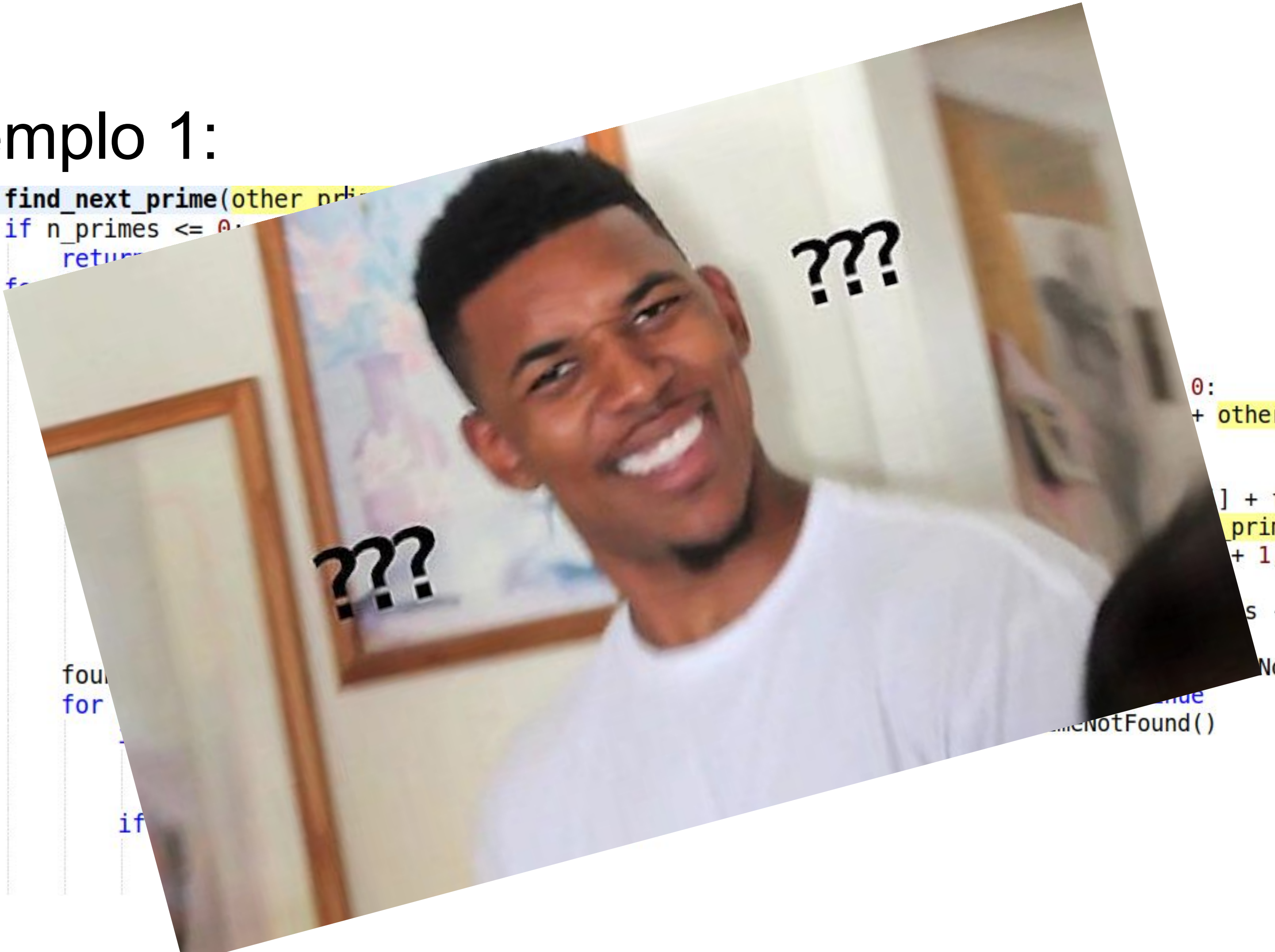


```python
        if found:
            if n_primes == 0:
                return [i] + other_primes
            else:
                try:
                    return [i] + find_next_prime(
                        other_primes + [i],
                        start + 1,
                        stop,
                        n_primes - 1
                    )
                except NextPrimeNotFound:
                    continue
    raise NextPrimeNotFound()
```

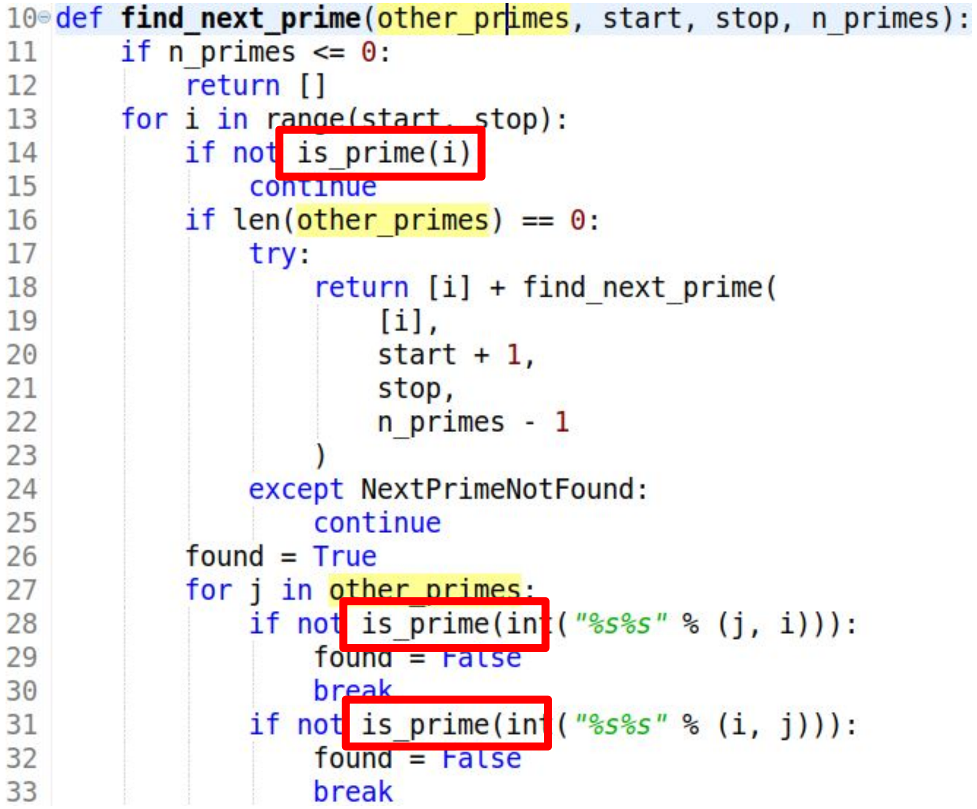
Exemplo 1:



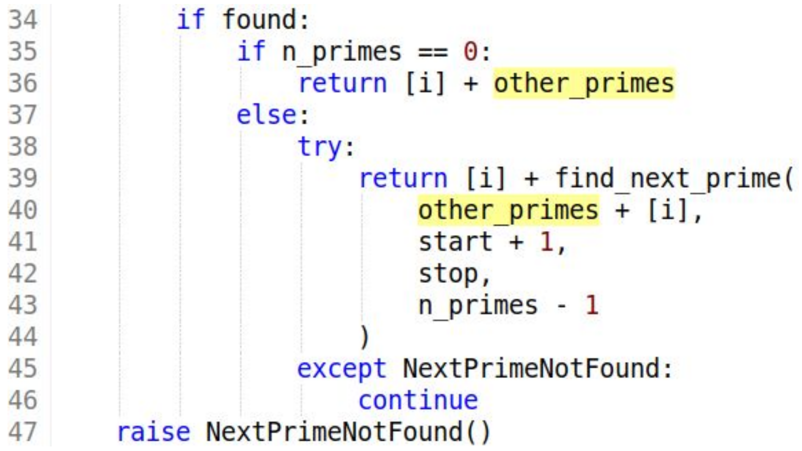
```
def find_next_prime(other_pr
    if n_primes <= 0:
        retur
    f



                                                                     0:
                                                                     + other_primes


                                                                      ] + find_next_prime(
                                                                      primes + [i],
                                                                      + 1,

                                                                      s - 1

        fou                                                           NotFound:
        for                                                          ue
                                                                   meNotFound()


            if

```



Exemplo 1:

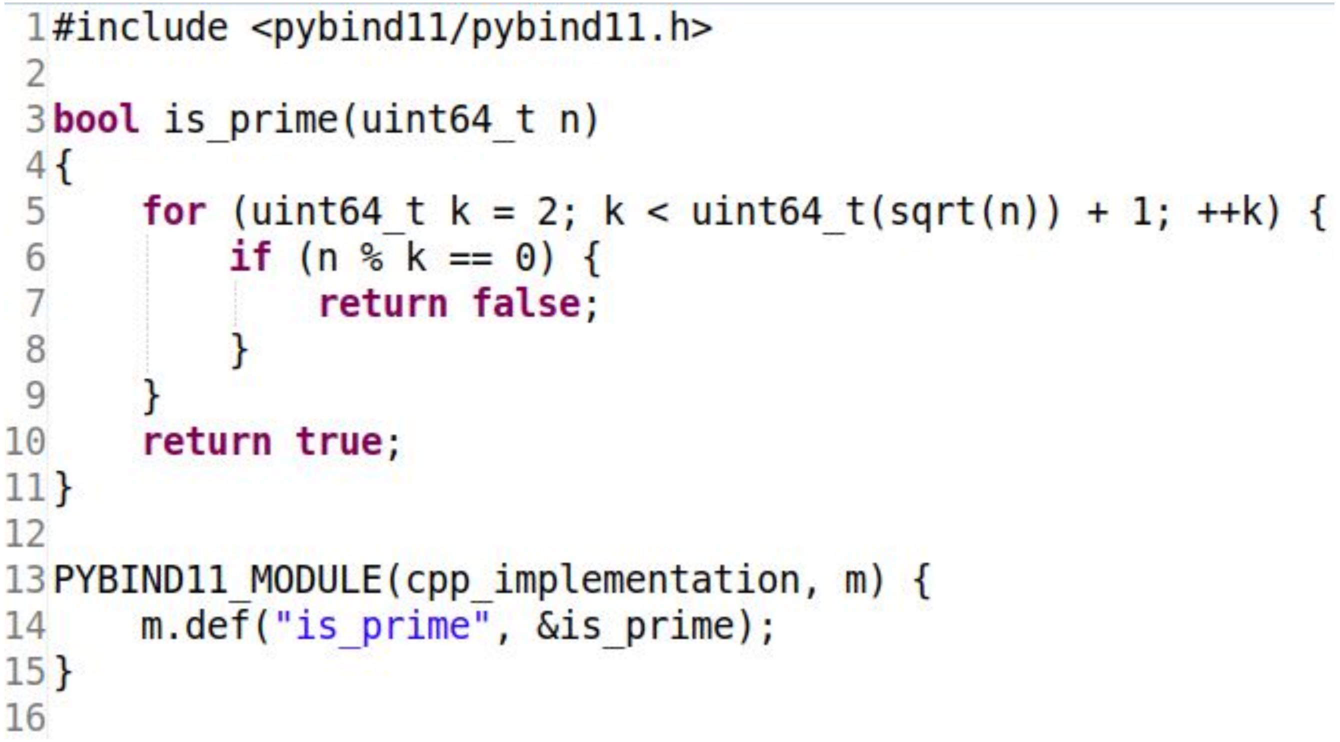


```python
def find_next_prime(other_primes, start, stop, n_primes):
    if n_primes <= 0:
        return []
    for i in range(start, stop):
        if not is_prime(i):
            continue
        if len(other_primes) == 0:
            try:
                return [i] + find_next_prime(
                    [i],
                    start + 1,
                    stop,
                    n_primes - 1
                )
            except NextPrimeNotFound:
                continue
        found = True
        for j in other_primes:
            if not is_prime(int("%s%s" % (j, i))):
                found = False
                break
            if not is_prime(int("%s%s" % (i, j))):
                found = False
                break
```



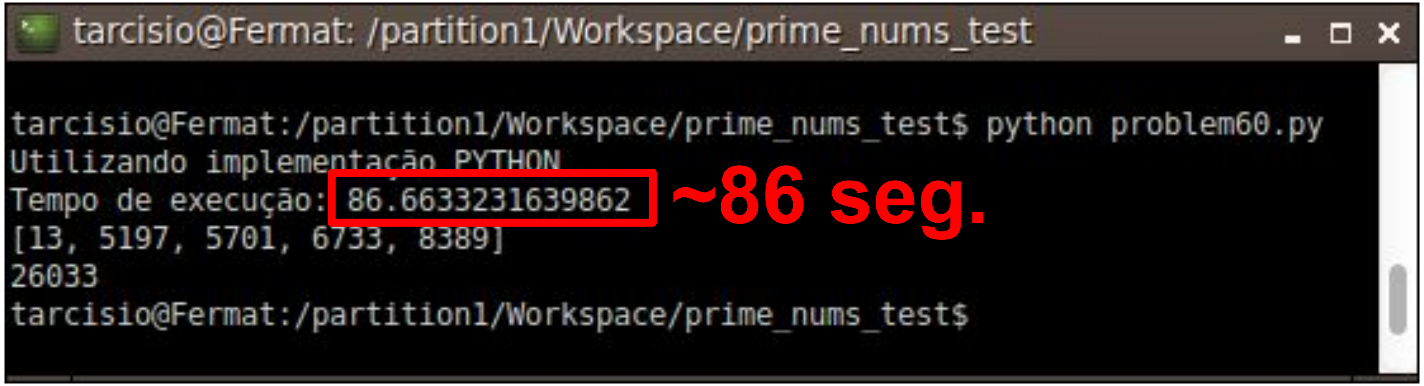
```python
        if found:
            if n_primes == 0:
                return [i] + other_primes
            else:
                try:
                    return [i] + find_next_prime(
                        other_primes + [i],
                        start + 1,
                        stop,
                        n_primes - 1
                    )
                except NextPrimeNotFound:
                    continue
    raise NextPrimeNotFound()
```

Exemplo 1:



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

bool is_prime(uint64_t n)
{
    for (uint64_t k = 2; k < uint64_t(sqrt(n)) + 1; ++k) {
        if (n % k == 0) {
            return false;
        }
    }
    return true;
}

PYBIND11_MODULE(cpp_implementation, m) {
    m.def("is_prime", &is_prime);
}

```

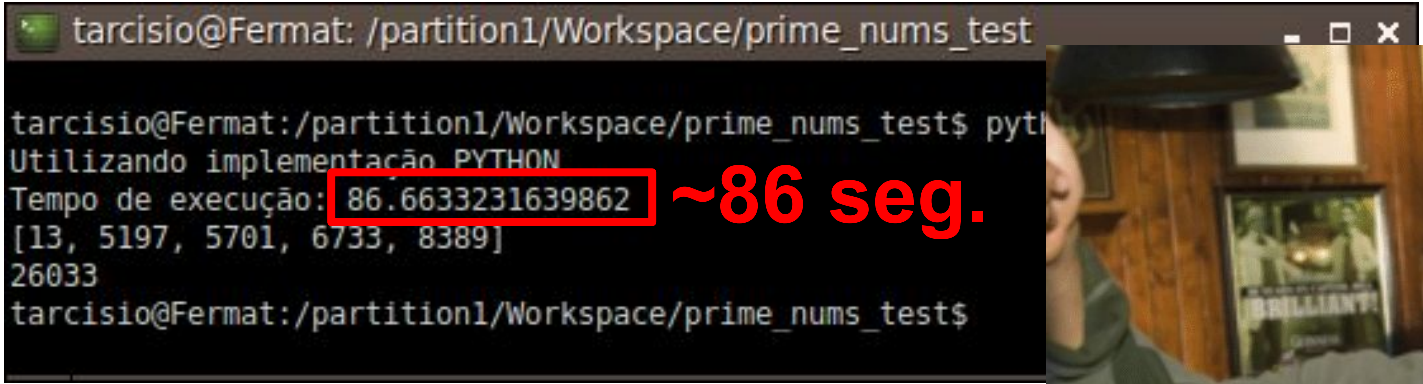
Exemplo 1:


tarcisio@Fermat: /partition1/Workspace/prime_nums_test

```
tarcisio@Fermat:/partition1/Workspace/prime_nums_test$ python problem60.py
Utilizando implementação PYTHON
Tempo de execução: 86.6633231639862
[13, 5197, 5701, 6733, 8389]
26033
tarcisio@Fermat:/partition1/Workspace/prime_nums_test$
```

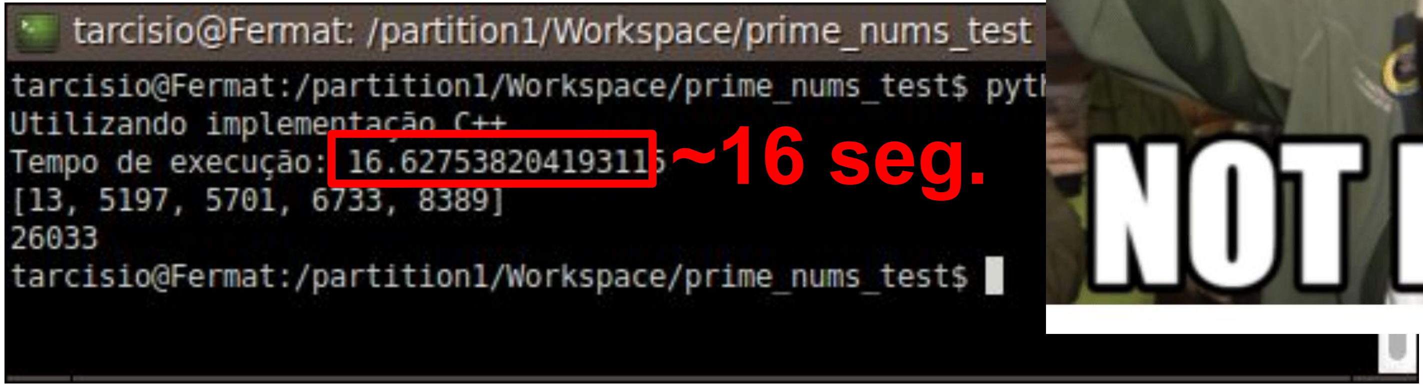
~86 seg.

Exemplo 1:



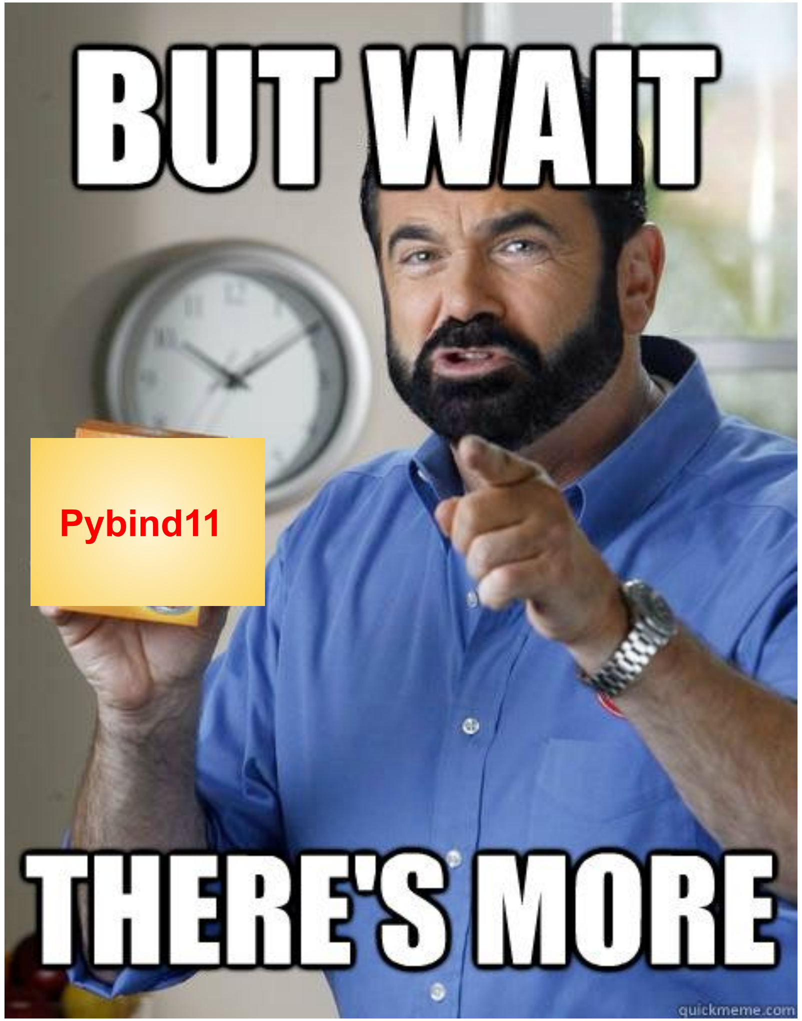
```
tarcisio@Fermat: /partition1/Workspace/prime_nums_test

tarcisio@Fermat:/partition1/Workspace/prime_nums_test$ pyt
Utilizando implementação PYTHON
Tempo de execução: 86.6633231639862
[13, 5197, 5701, 6733, 8389]
26033
tarcisio@Fermat:/partition1/Workspace/prime_nums_test$
```

~86 seg.

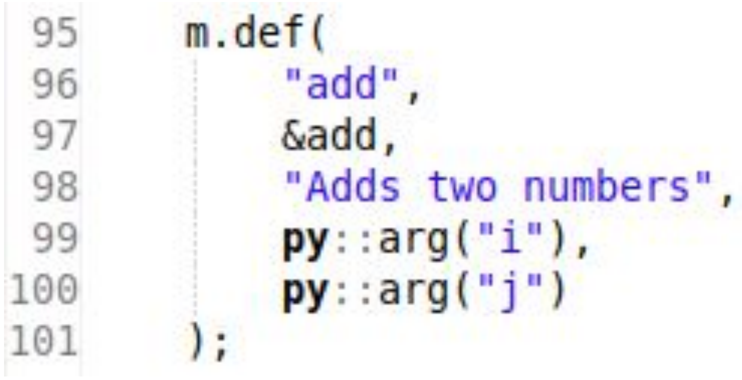


```
tarcisio@Fermat: /partition1/Workspace/prime_nums_test
tarcisio@Fermat:/partition1/Workspace/prime_nums_test$ pyt
Utilizando implementação C++
Tempo de execução: 16.627538204193115
[13, 5197, 5701, 6733, 8389]
26033
tarcisio@Fermat:/partition1/Workspace/prime_nums_test$
```

~16 seg.

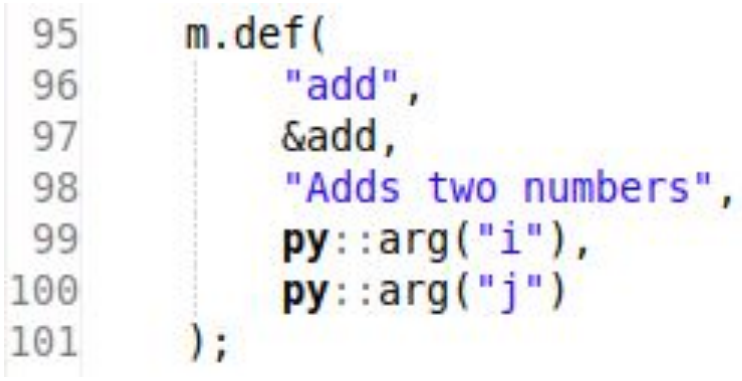

BUT WAIT
Pybind11
THERE'S MORE
quickmeme.com

Documentação & argumentos

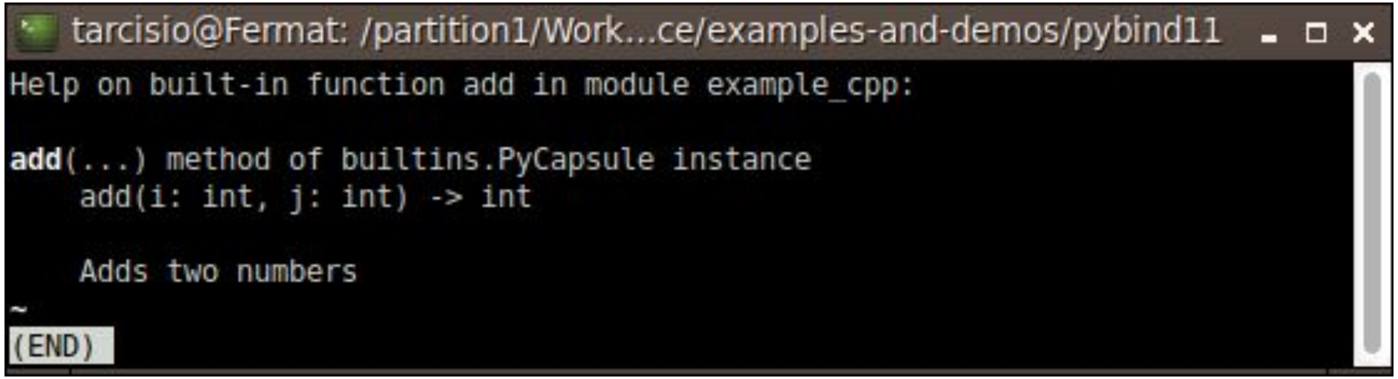


```
m.def(
    "add",
    &add,
    "Adds two numbers",
    py::arg("i"),
    py::arg("j")
);
```

Documentação & argumentos



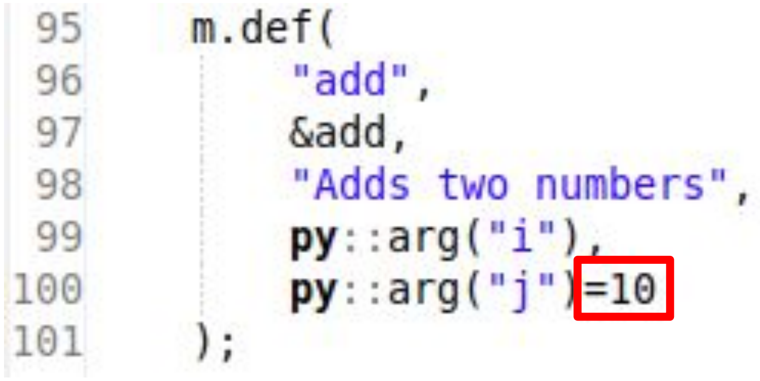
```cpp
    m.def(
        "add",
        &add,
        "Adds two numbers",
        py::arg("i"),
        py::arg("j")
    );
```



tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

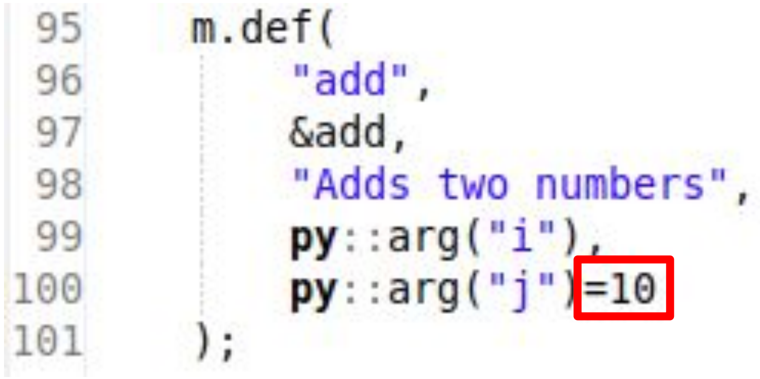
```
Help on built-in function add in module example_cpp:

add(...) method of builtins.PyCapsule instance
    add(i: int, j: int) -> int

    Adds two numbers
~
(END)
```

Parâmetros default

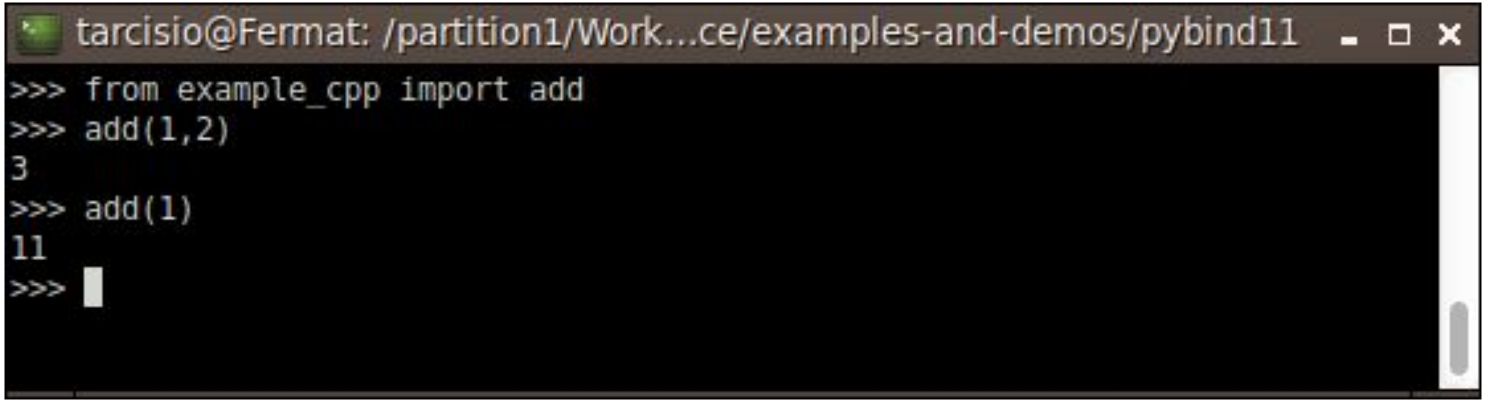


```
m.def(
    "add",
    &add,
    "Adds two numbers",
    py::arg("i"),
    py::arg("j")=10
);
```

Parâmetros default



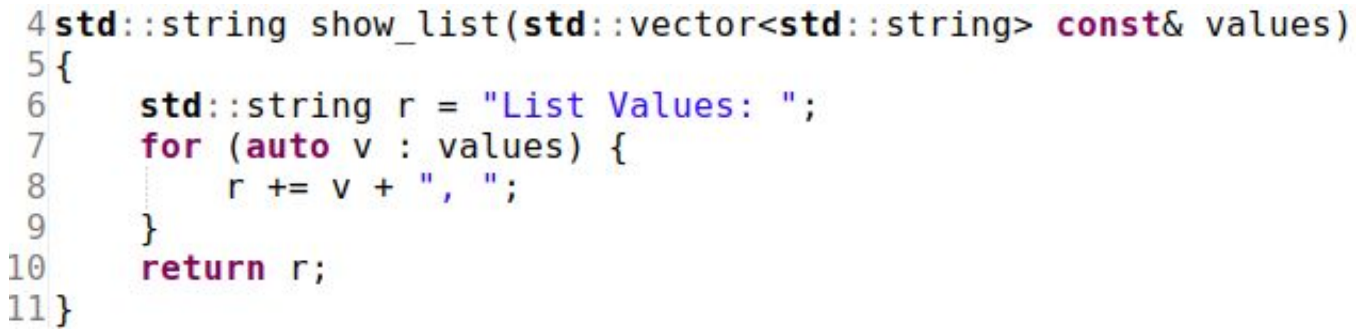
```
    m.def(
        "add",
        &add,
        "Adds two numbers",
        py::arg("i"),
        py::arg("j")=10
    );
```



tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

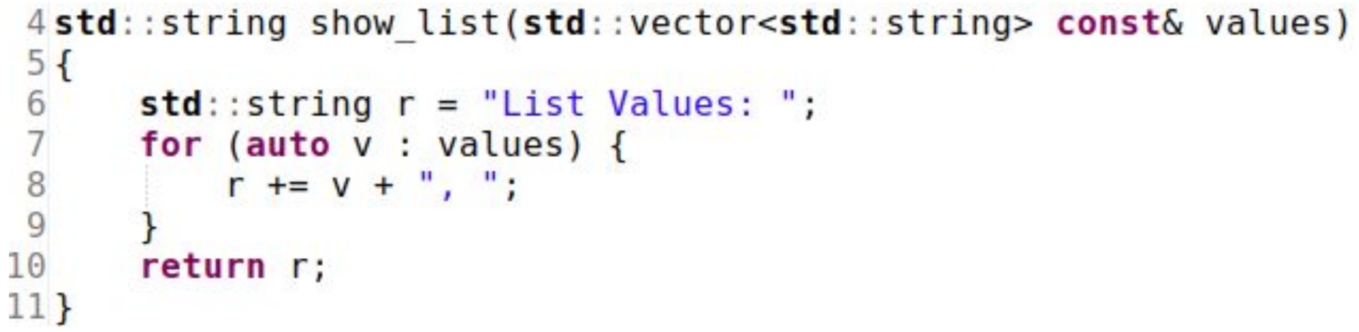
```
>>> from example_cpp import add
>>> add(1,2)
3
>>> add(1)
11
>>> 
```

Tipos Primitivos (C++)

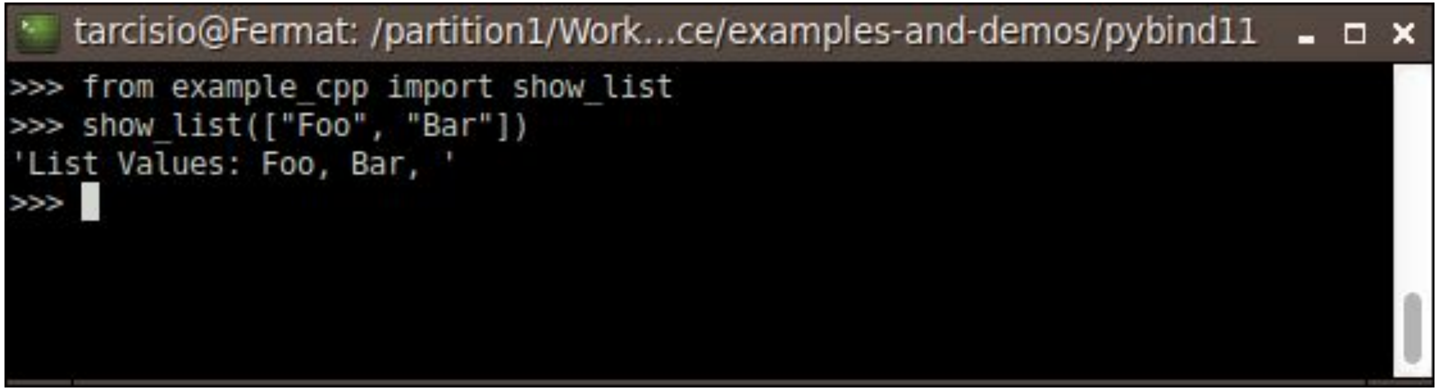


```cpp
std::string show_list(std::vector<std::string> const& values)
{
    std::string r = "List Values: ";
    for (auto v : values) {
        r += v + ", ";
    }
    return r;
}
```

Tipos Primitivos (C++)



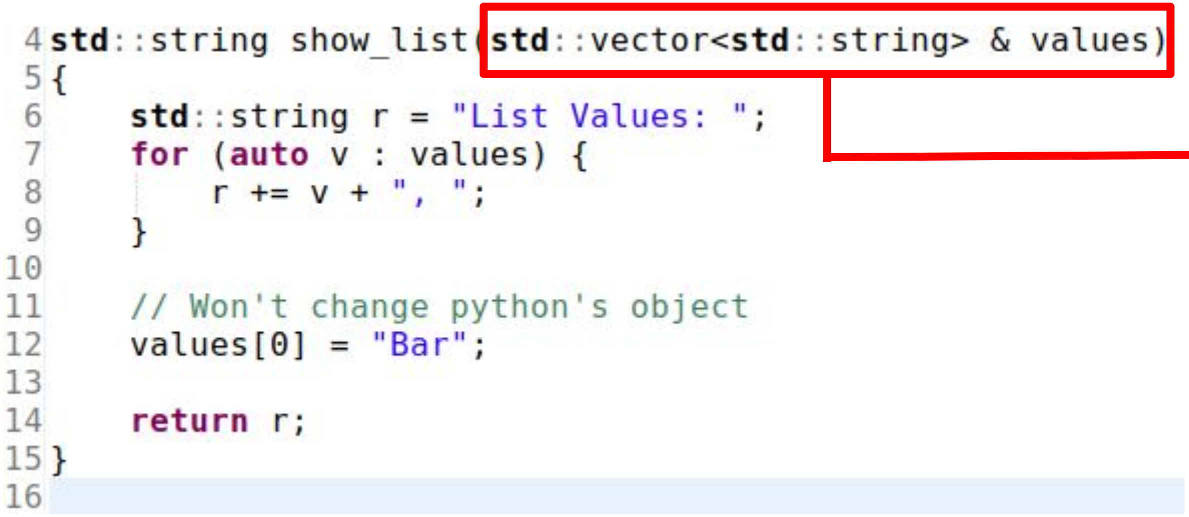
```cpp
std::string show_list(std::vector<std::string> const& values)
{
    std::string r = "List Values: ";
    for (auto v : values) {
        r += v + ", ";
    }
    return r;
}
```



tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

```
>>> from example_cpp import show_list
>>> show_list(["Foo", "Bar"])
'List Values: Foo, Bar, '
>>> 
```

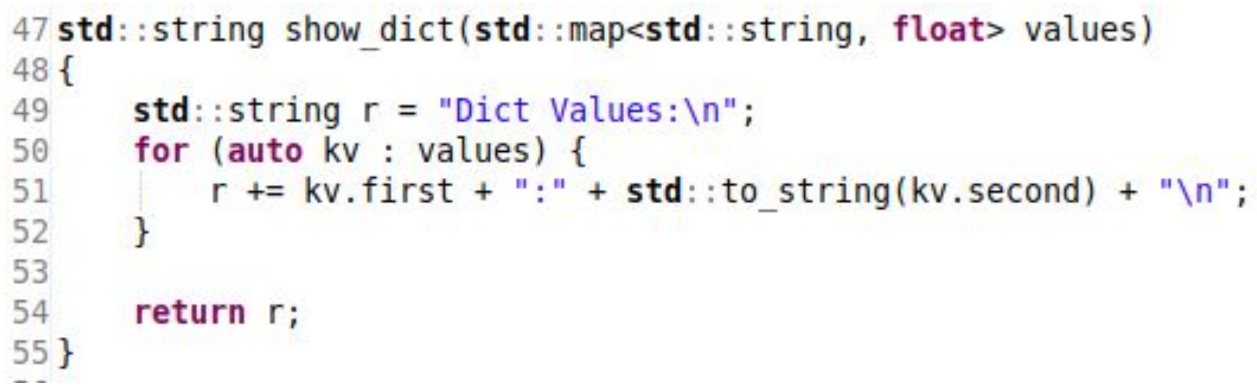
Tipos Primitivos (C++)



```
std::string show_list(std::vector<std::string> & values)
{
    std::string r = "List Values: ";
    for (auto v : values) {
        r += v + ", ";
    }

    // Won't change python's object
    values[0] = "Bar";

    return r;
}

```

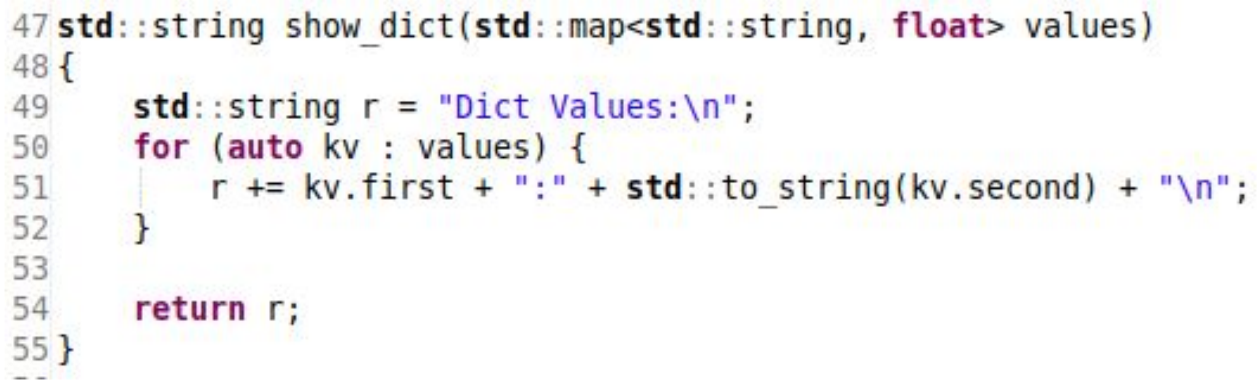
Apesar de aparecer como uma referência, o conversor do pybind11, na verdade, faz uma cópia de todos os dados antes de passa-los para uma função C++.

Tipos Primitivos (C++)

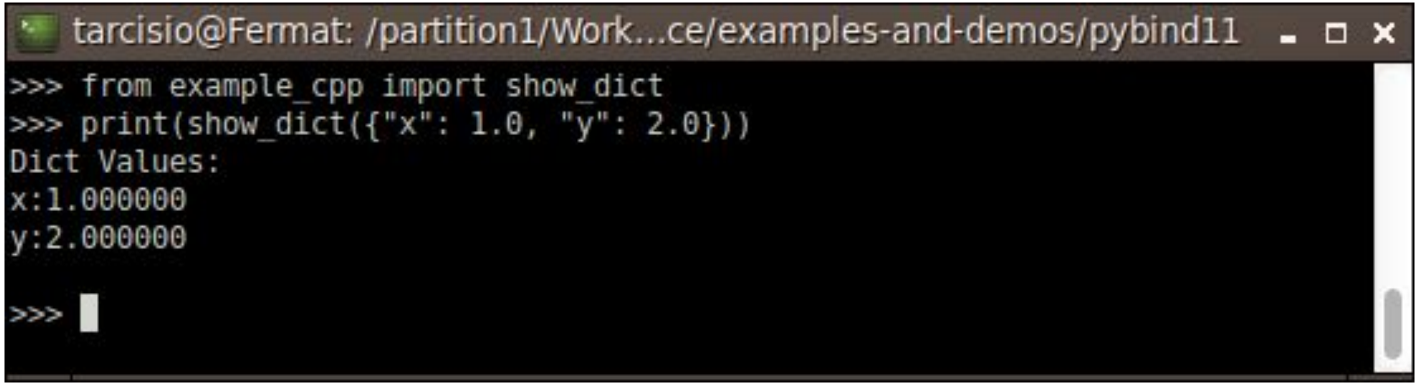


```cpp
std::string show_dict(std::map<std::string, float> values)
{
    std::string r = "Dict Values:\n";
    for (auto kv : values) {
        r += kv.first + ":" + std::to_string(kv.second) + "\n";
    }

    return r;
}
```

Tipos Primitivos (C++)



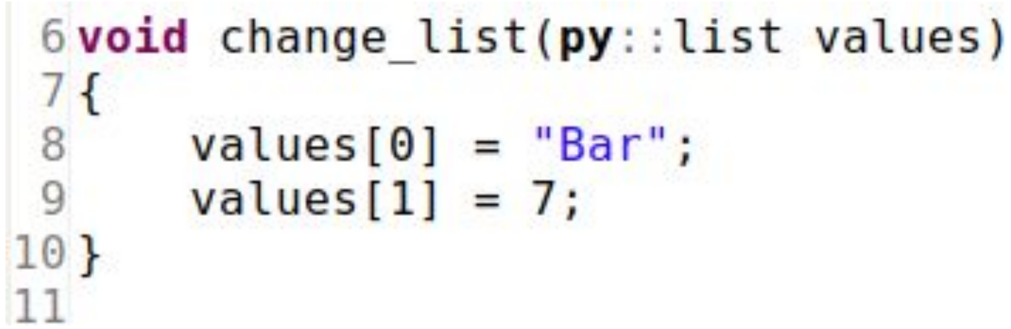
```cpp
std::string show_dict(std::map<std::string, float> values)
{
    std::string r = "Dict Values:\n";
    for (auto kv : values) {
        r += kv.first + ":" + std::to_string(kv.second) + "\n";
    }

    return r;
}
```



tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

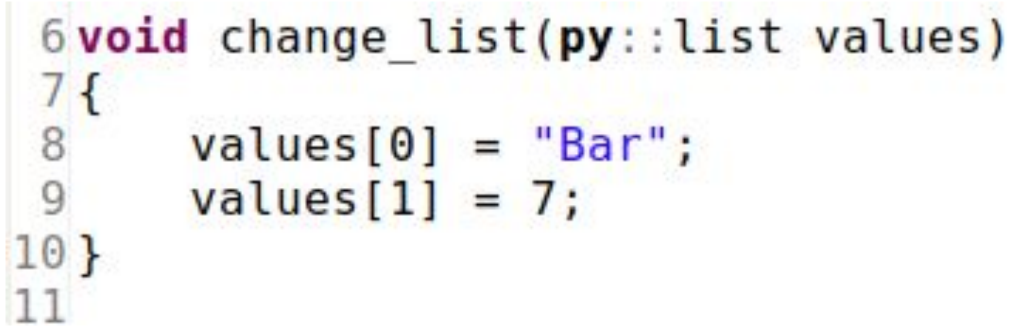
```
>>> from example_cpp import show_dict
>>> print(show_dict({"x": 1.0, "y": 2.0}))
Dict Values:
x:1.000000
y:2.000000

>>>
```

Tipos Primitivos (Python)

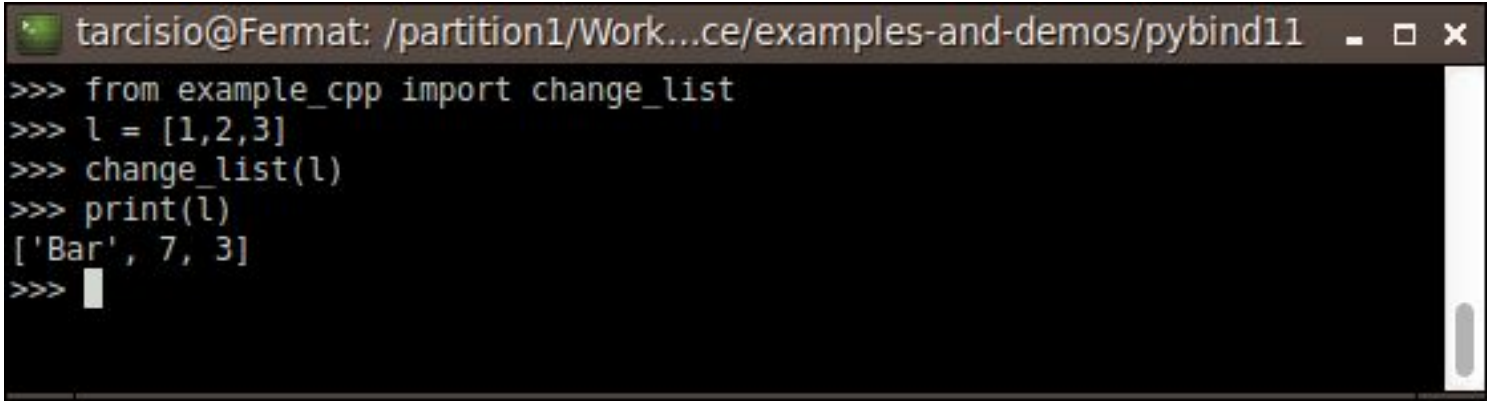


```
void change_list(py::list values)
{
    values[0] = "Bar";
    values[1] = 7;
}

```

Tipos Primitivos (Python)



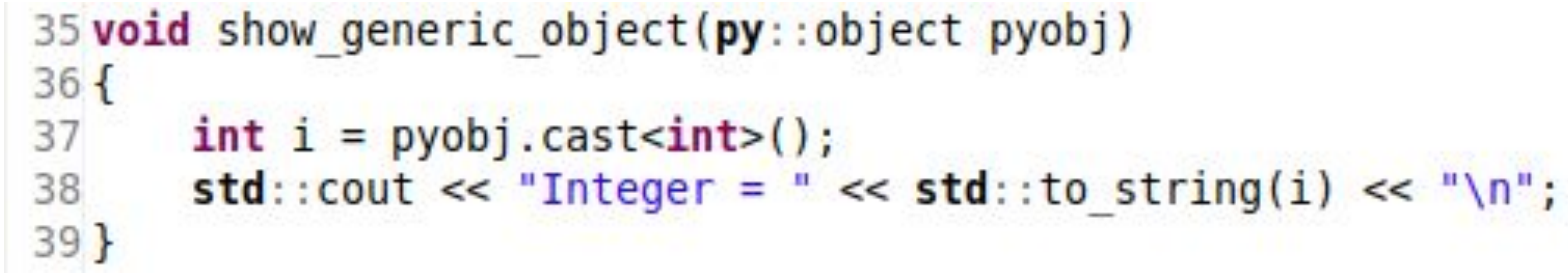
```cpp
void change_list(py::list values)
{
    values[0] = "Bar";
    values[1] = 7;
}

```



tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

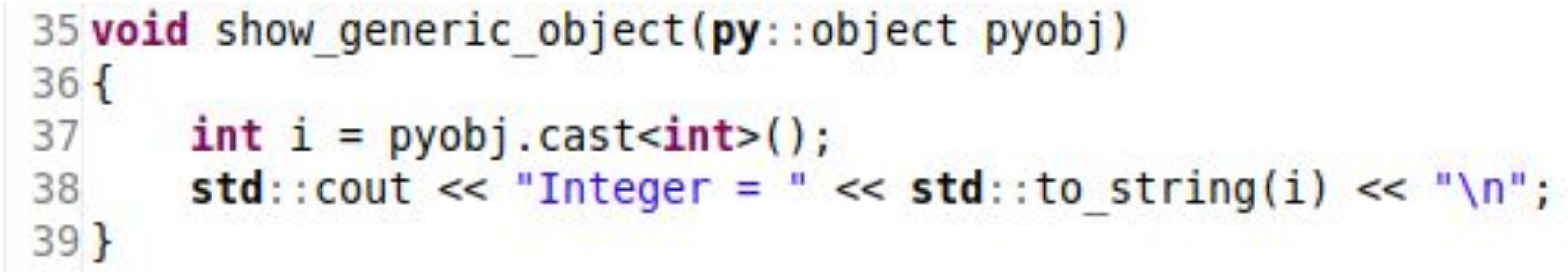
```
>>> from example_cpp import change_list
>>> l = [1,2,3]
>>> change_list(l)
>>> print(l)
['Bar', 7, 3]
>>> 
```

Tipos Primitivos (Python)

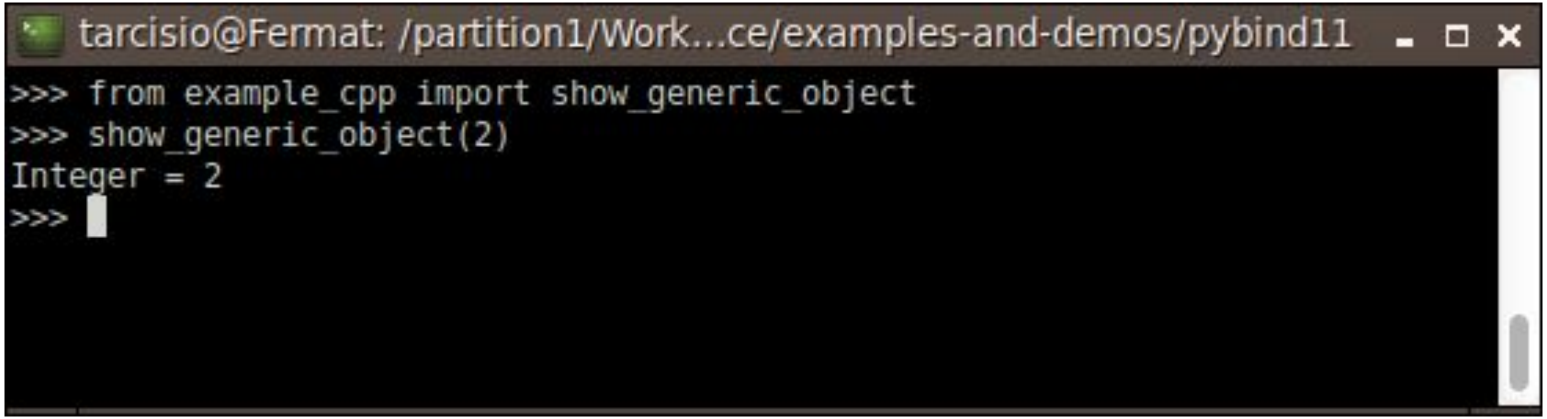


```
void show_generic_object(py::object pyobj)
{
    int i = pyobj.cast<int>();
    std::cout << "Integer = " << std::to_string(i) << "\n";
}
```

Tipos Primitivos (Python)



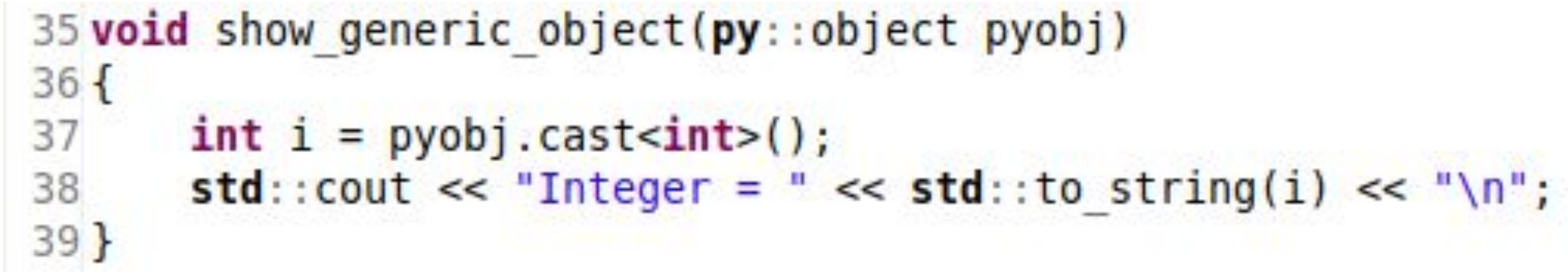
```cpp
void show_generic_object(py::object pyobj)
{
    int i = pyobj.cast<int>();
    std::cout << "Integer = " << std::to_string(i) << "\n";
}
```



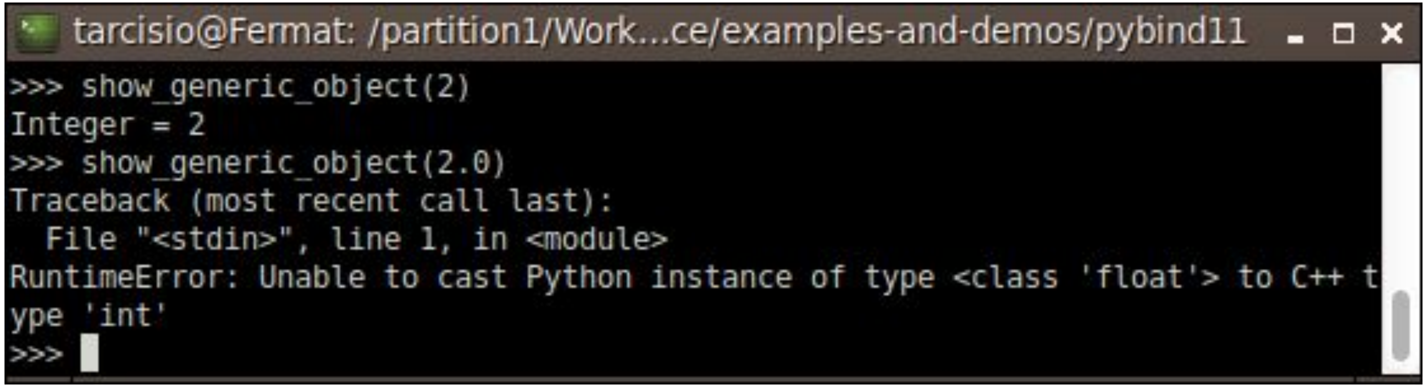
tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

```
>>> from example_cpp import show_generic_object
>>> show_generic_object(2)
Integer = 2
>>> 
```

Conversão automática de exceções



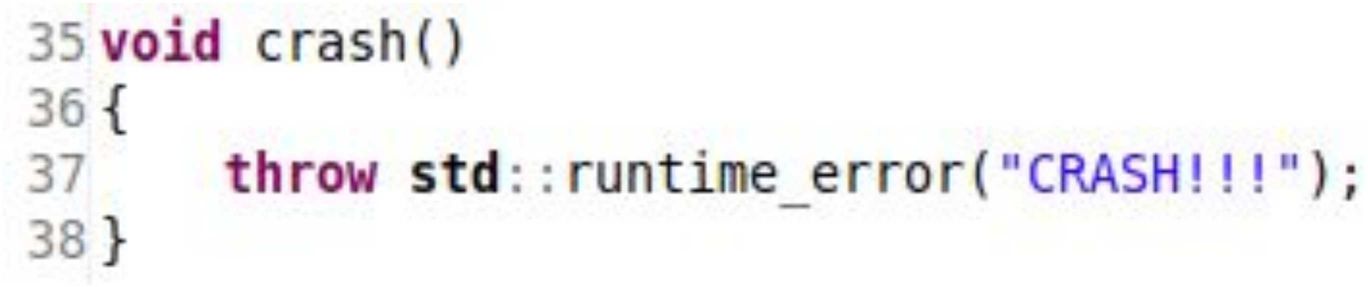
```cpp
void show_generic_object(py::object pyobj)
{
    int i = pyobj.cast<int>();
    std::cout << "Integer = " << std::to_string(i) << "\n";
}
```



tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

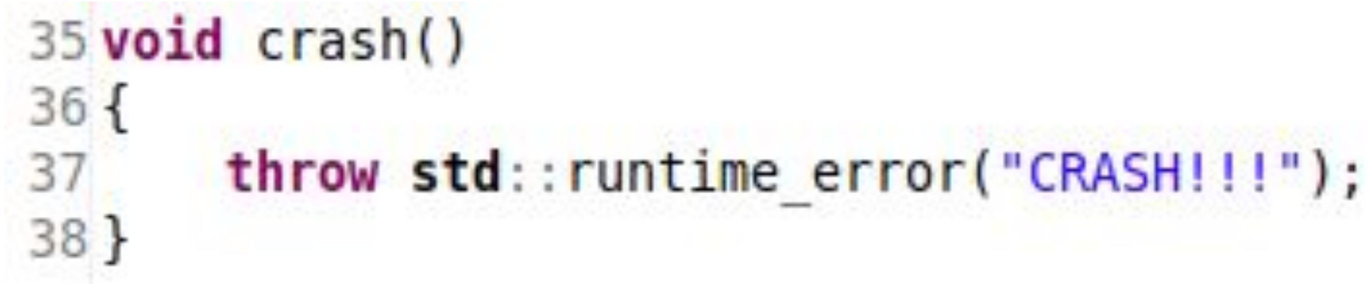
```
>>> show_generic_object(2)
Integer = 2
>>> show_generic_object(2.0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: Unable to cast Python instance of type <class 'float'> to C++ t
ype 'int'
>>> 
```

Conversão automática de exceções

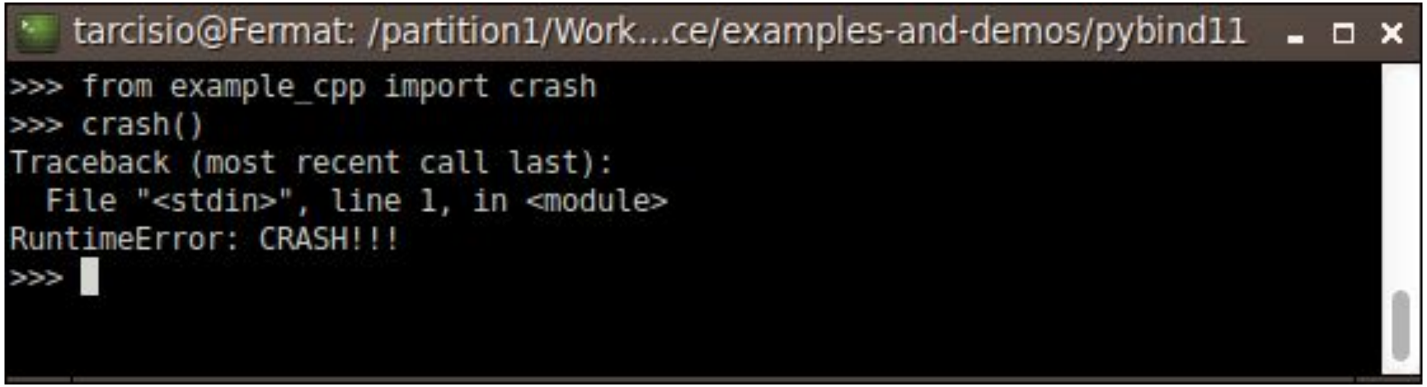


```cpp
void crash()
{
    throw std::runtime_error("CRASH!!!");
}
```

Conversão automática de exceções



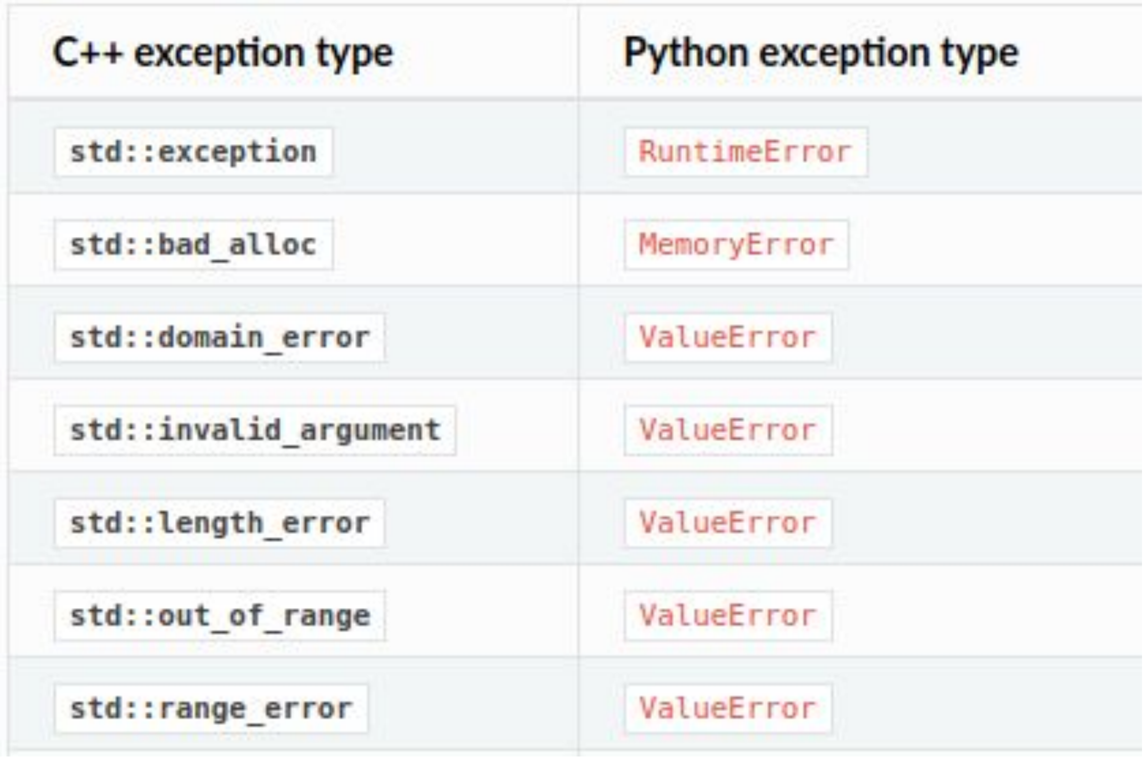
```cpp
void crash()
{
    throw std::runtime_error("CRASH!!!");
}
```



tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

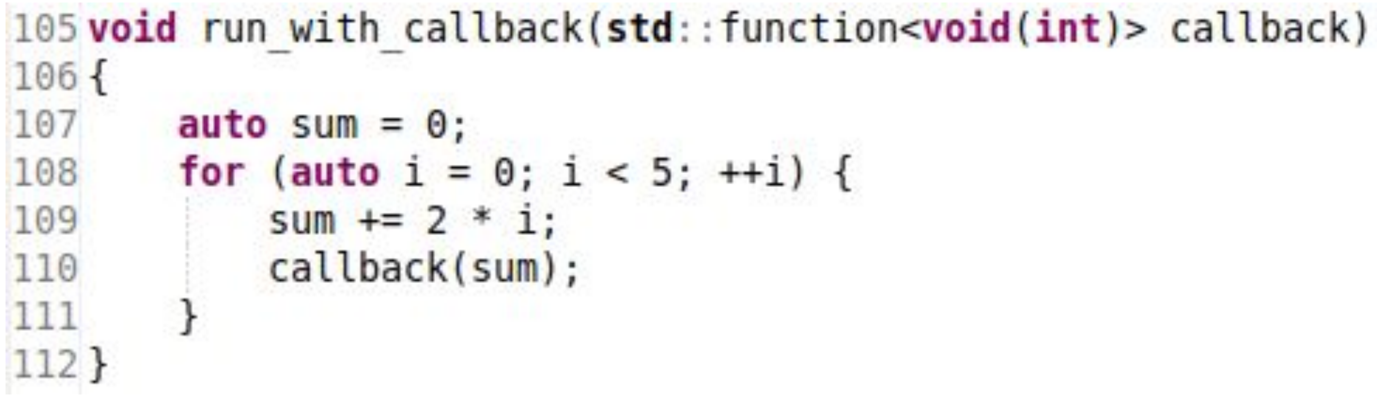
```
>>> from example_cpp import crash
>>> crash()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: CRASH!!!
>>> 
```

Conversão automática de exceções



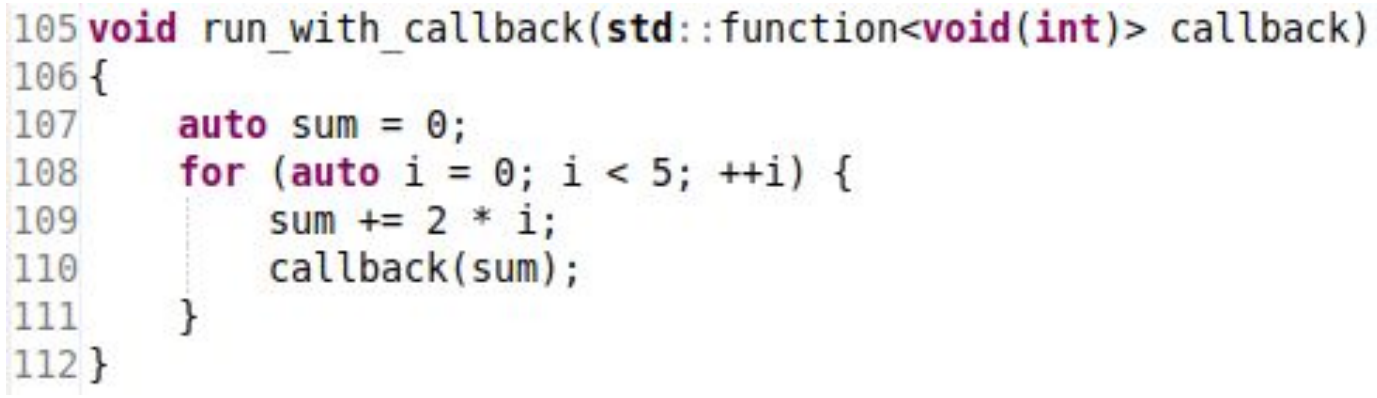
C++ exception type	Python exception type
`std::exception`	`RuntimeError`
`std::bad_alloc`	`MemoryError`
`std::domain_error`	`ValueError`
`std::invalid_argument`	`ValueError`
`std::length_error`	`ValueError`
`std::out_of_range`	`ValueError`
`std::range_error`	`ValueError`

Funções como argumento

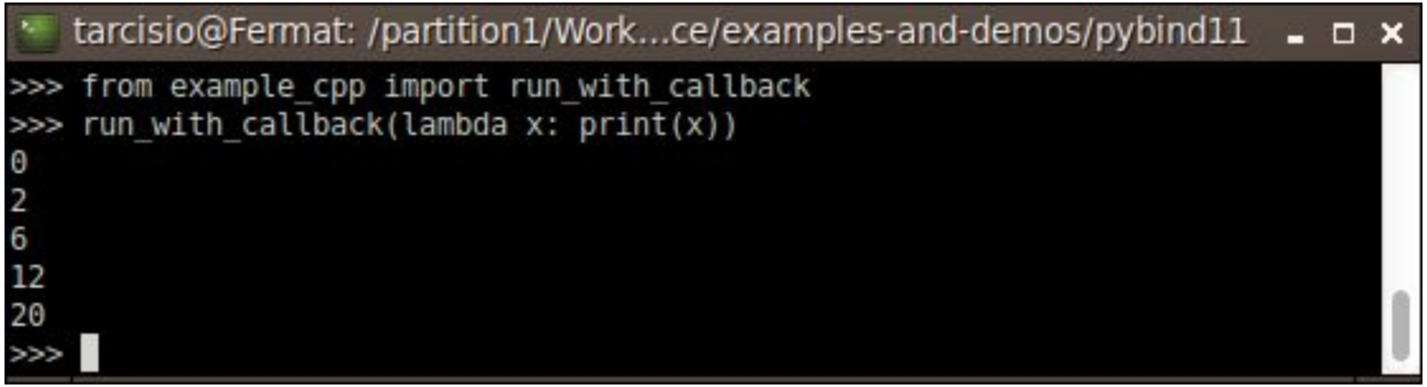


```cpp
void run_with_callback(std::function<void(int)> callback)
{
    auto sum = 0;
    for (auto i = 0; i < 5; ++i) {
        sum += 2 * i;
        callback(sum);
    }
}
```

Funções como argumento



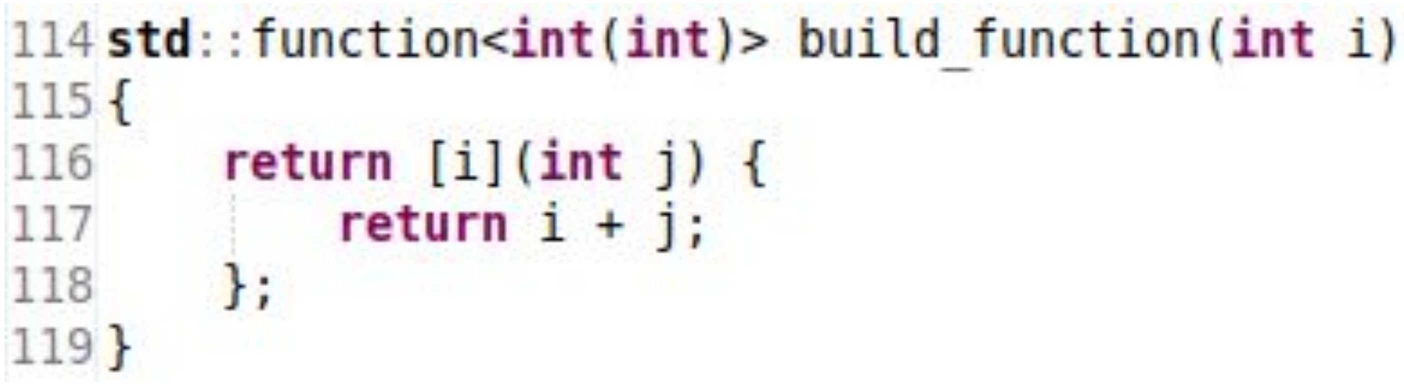
```cpp
void run_with_callback(std::function<void(int)> callback)
{
    auto sum = 0;
    for (auto i = 0; i < 5; ++i) {
        sum += 2 * i;
        callback(sum);
    }
}
```



tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

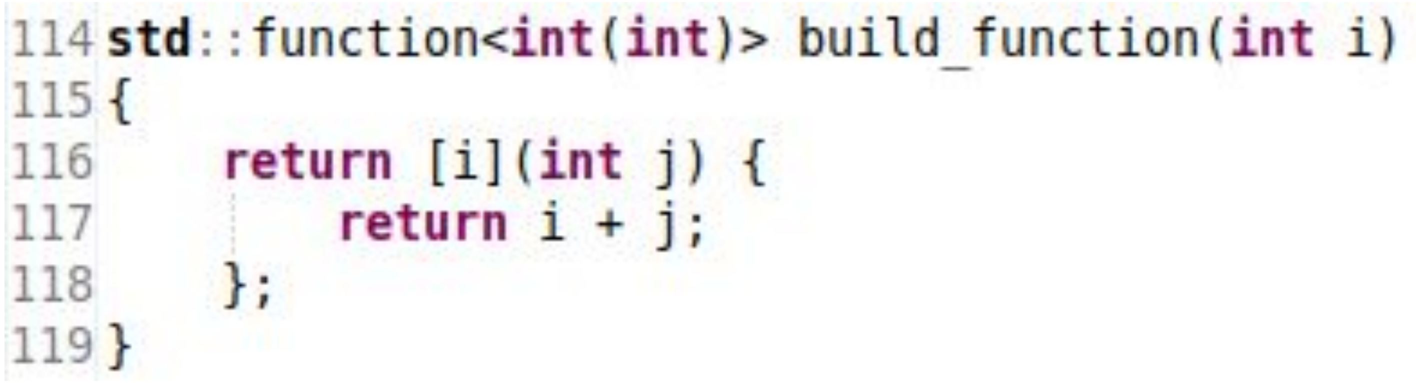
```
>>> from example_cpp import run_with_callback
>>> run_with_callback(lambda x: print(x))
0
2
6
12
20
>>> 
```

Funções como argumento

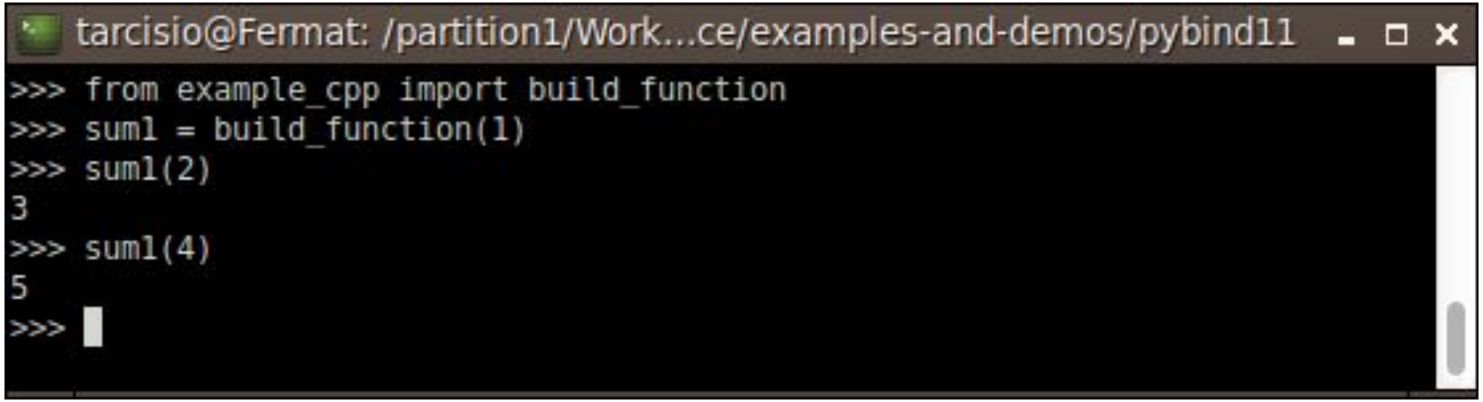


```
std::function<int(int)> build_function(int i)
{
    return [i](int j) {
        return i + j;
    };
}
```

Funções como argumento



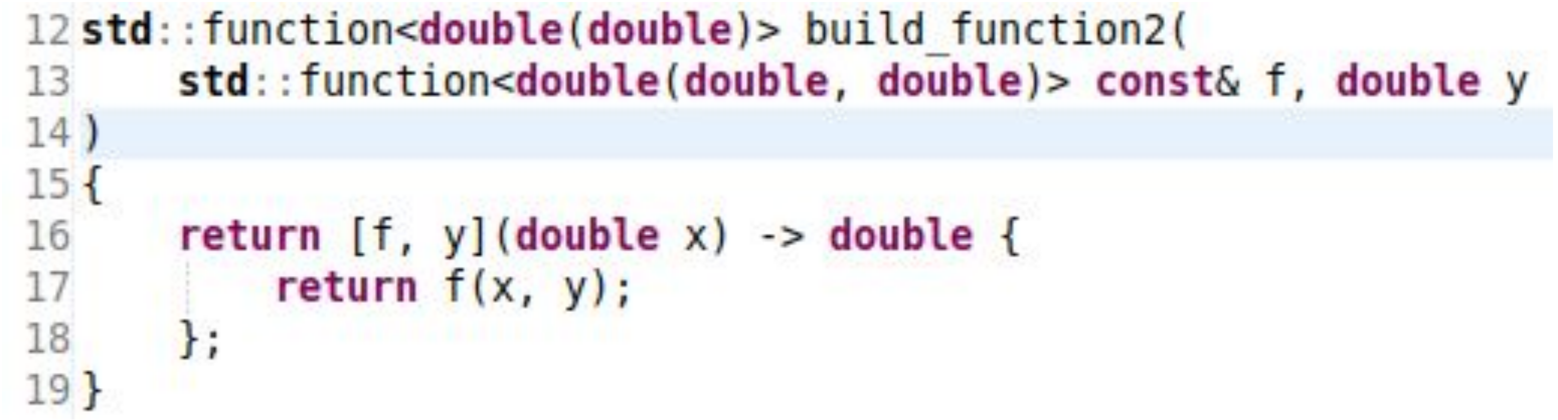
```cpp
std::function<int(int)> build_function(int i)
{
    return [i](int j) {
        return i + j;
    };
}
```



tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

```
>>> from example_cpp import build_function
>>> sum1 = build_function(1)
>>> sum1(2)
3
>>> sum1(4)
5
>>>
```

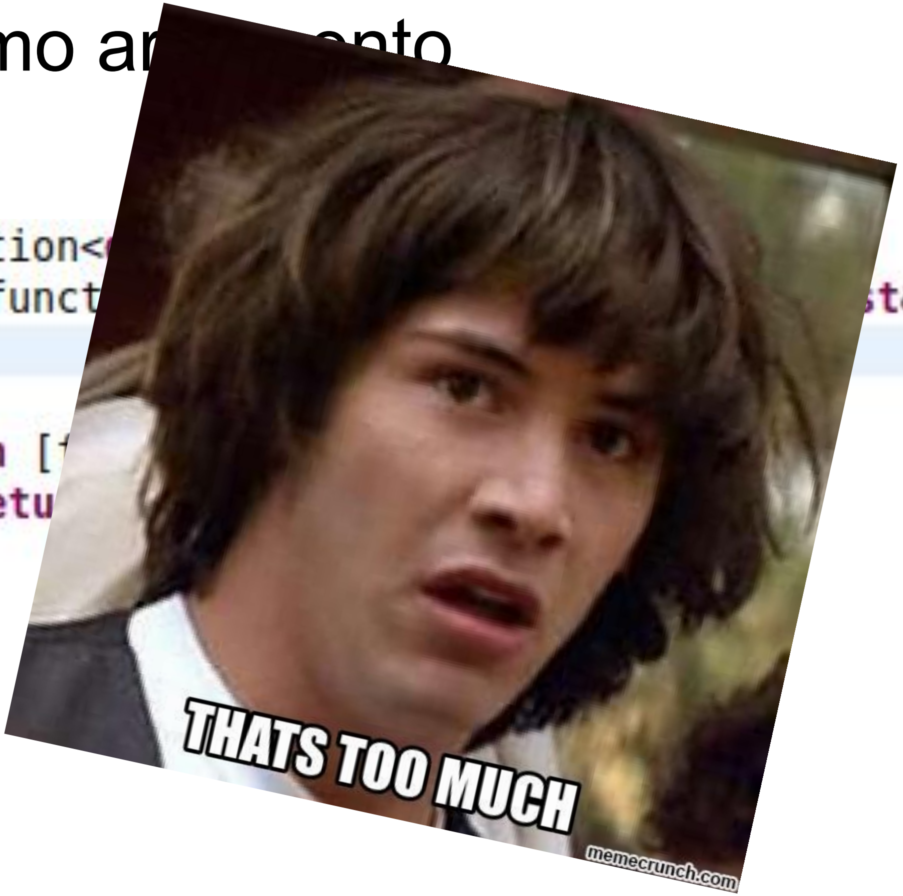
Funções como argumento



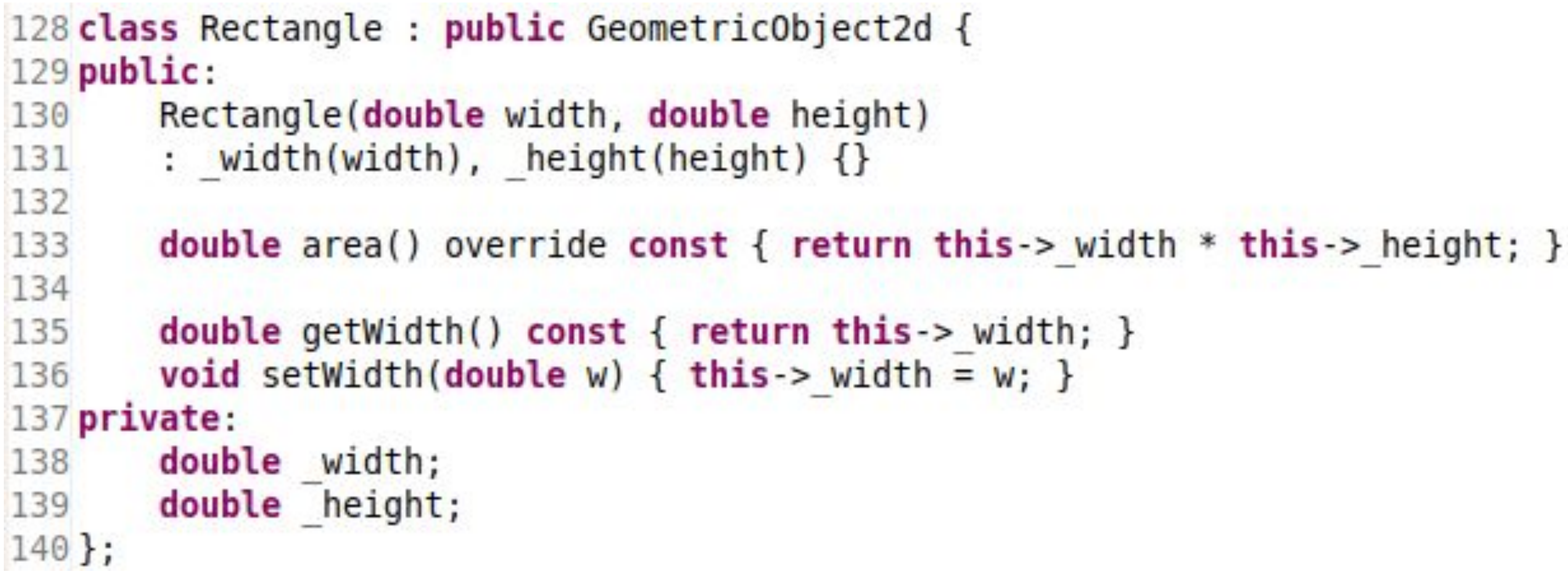
```cpp
std::function<double(double)> build_function2(
    std::function<double(double, double)> const& f, double y
)
{
    return [f, y](double x) -> double {
        return f(x, y);
    };
}
```

Funções como ar

```
std::function<
    std::funct                                        t& f, double y
)
{
    return [f
        retu
    };
}
```

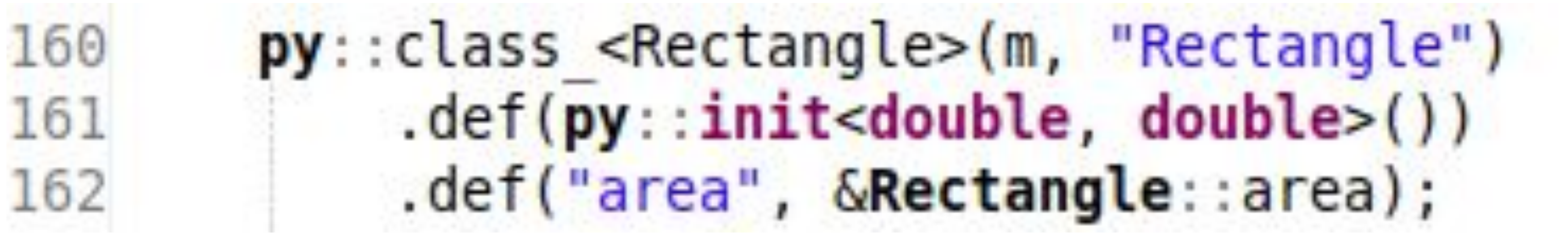


Classes & Objetos



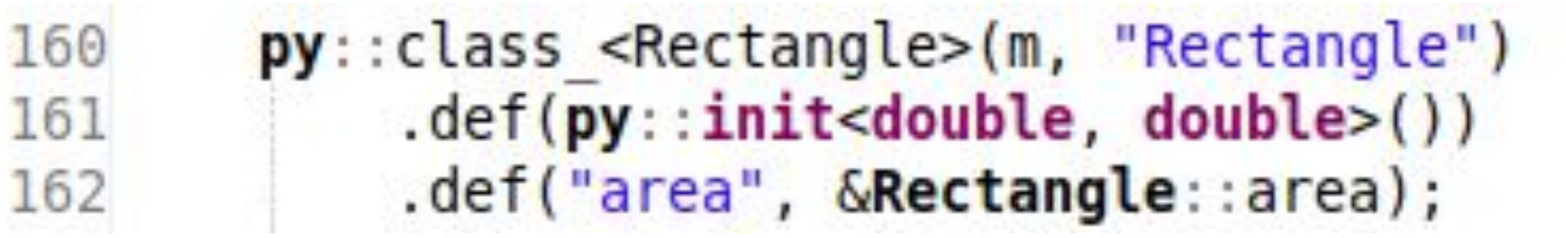
```
class Rectangle : public GeometricObject2d {
public:
    Rectangle(double width, double height)
    : _width(width), _height(height) {}

    double area() override const { return this->_width * this->_height; }

    double getWidth() const { return this->_width; }
    void setWidth(double w) { this->_width = w; }
private:
    double _width;
    double _height;
};
```

Classes & Objetos

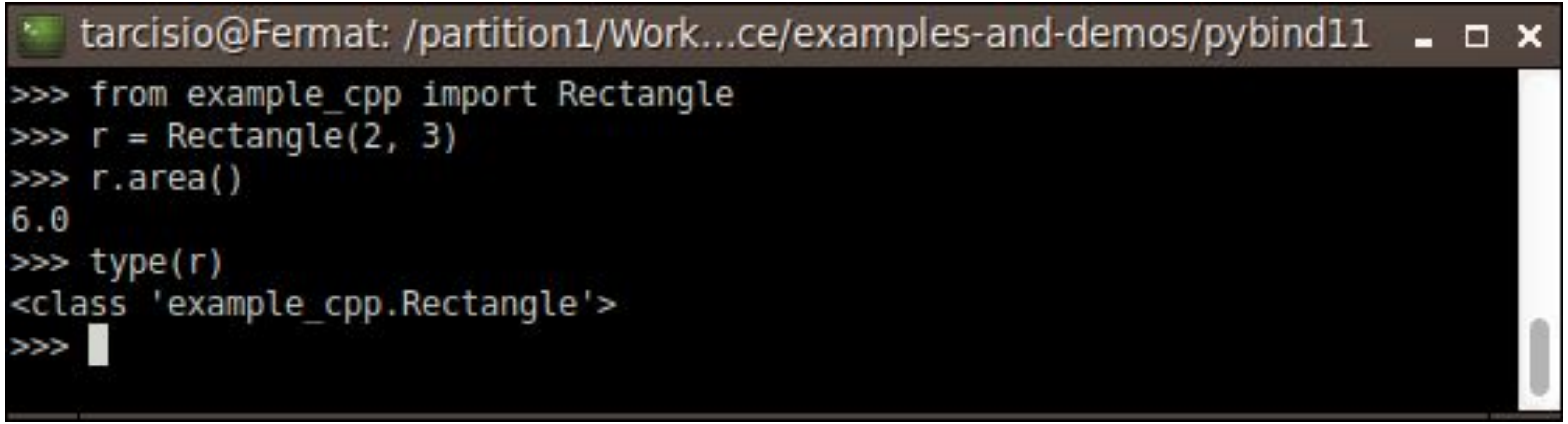


```
    py::class_<Rectangle>(m, "Rectangle")
        .def(py::init<double, double>())
        .def("area", &Rectangle::area);
```

Classes & Objetos

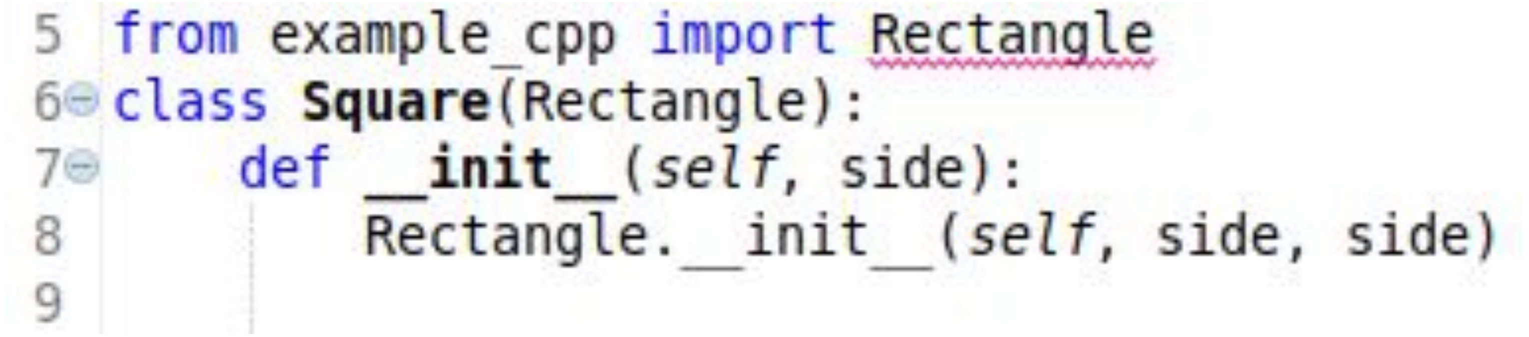


```
    py::class_<Rectangle>(m, "Rectangle")
        .def(py::init<double, double>())
        .def("area", &Rectangle::area);
```


tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

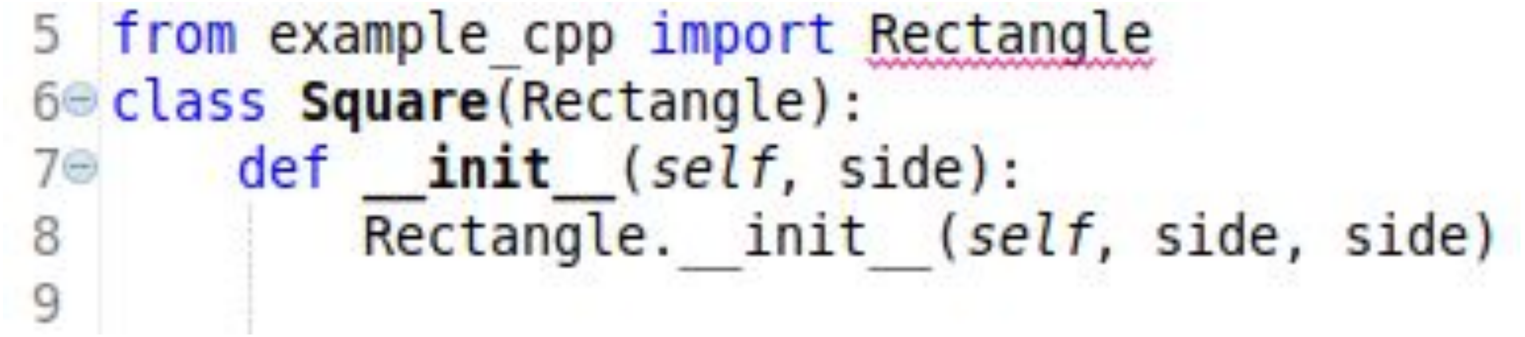
```
>>> from example_cpp import Rectangle
>>> r = Rectangle(2, 3)
>>> r.area()
6.0
>>> type(r)
<class 'example_cpp.Rectangle'>
>>> 
```

Classes & Objetos

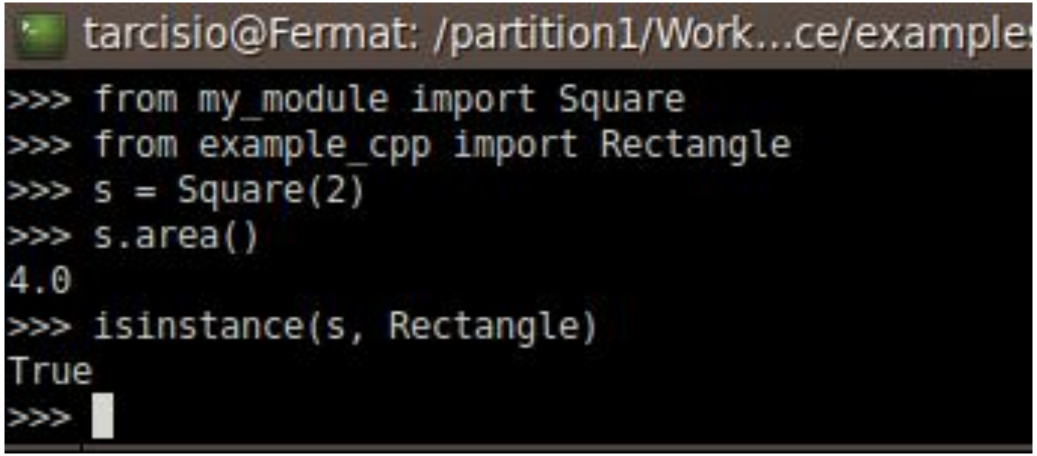


```python
from example_cpp import Rectangle
class Square(Rectangle):
    def __init__(self, side):
        Rectangle.__init__(self, side, side)

```

Classes & Objetos



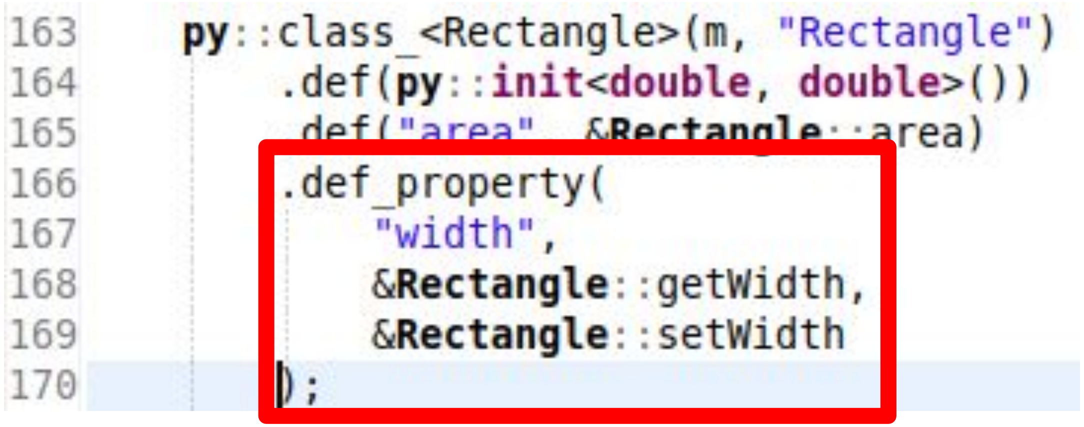
```python
from example_cpp import Rectangle
class Square(Rectangle):
    def __init__(self, side):
        Rectangle.__init__(self, side, side)

```



tarcisio@Fermat: /partition1/Work...ce/example

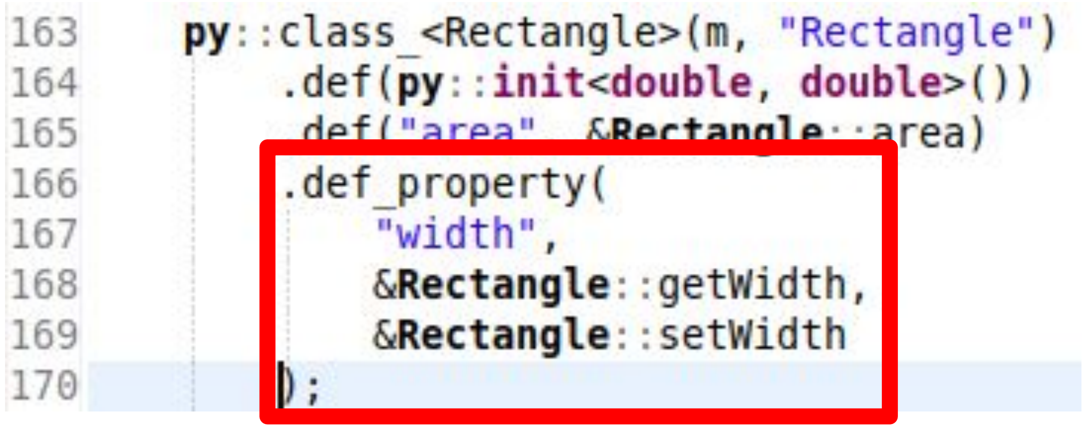
```
>>> from my_module import Square
>>> from example_cpp import Rectangle
>>> s = Square(2)
>>> s.area()
4.0
>>> isinstance(s, Rectangle)
True
>>> 
```

Classes & Objetos

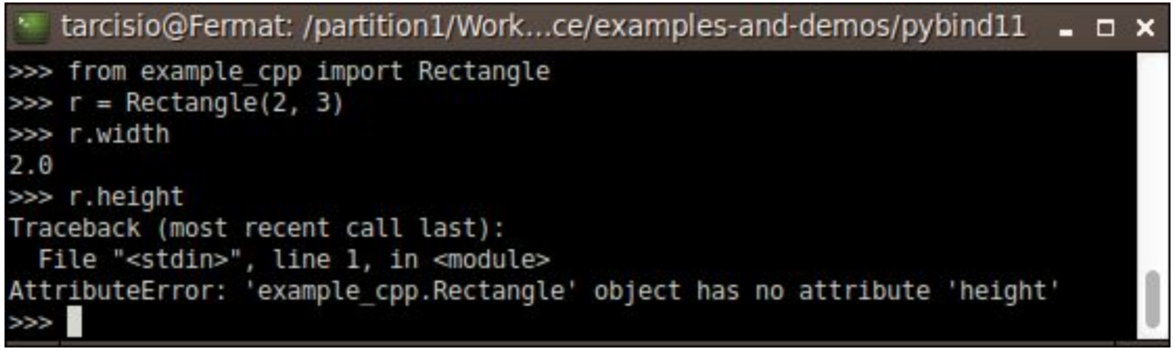


```
    py::class_<Rectangle>(m, "Rectangle")
        .def(py::init<double, double>())
        .def("area", &Rectangle::area)
        .def_property(
            "width",
            &Rectangle::getWidth,
            &Rectangle::setWidth
        );
```

Classes & Objetos



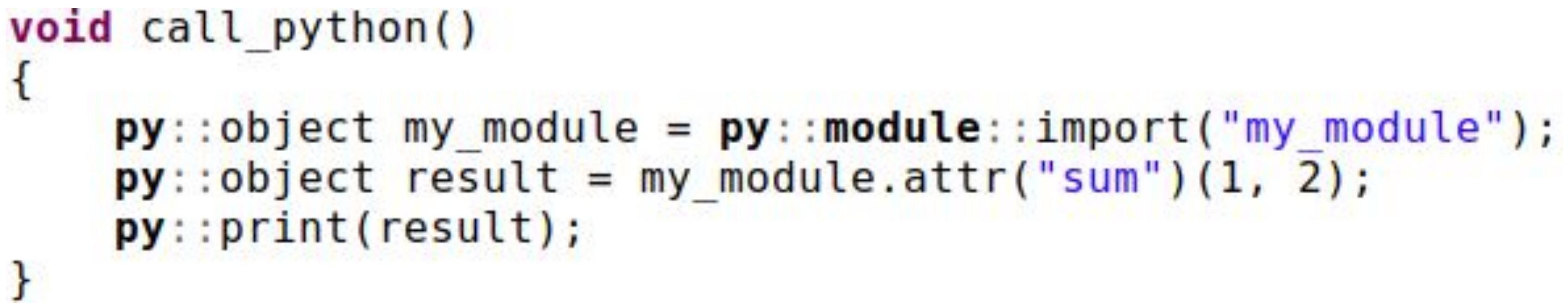
```
    py::class_<Rectangle>(m, "Rectangle")
        .def(py::init<double, double>())
        .def("area", &Rectangle::area)
        .def_property(
            "width",
            &Rectangle::getWidth,
            &Rectangle::setWidth
        );
```



tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

```
>>> from example_cpp import Rectangle
>>> r = Rectangle(2, 3)
>>> r.width
2.0
>>> r.height
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'example_cpp.Rectangle' object has no attribute 'height'
>>> 
```

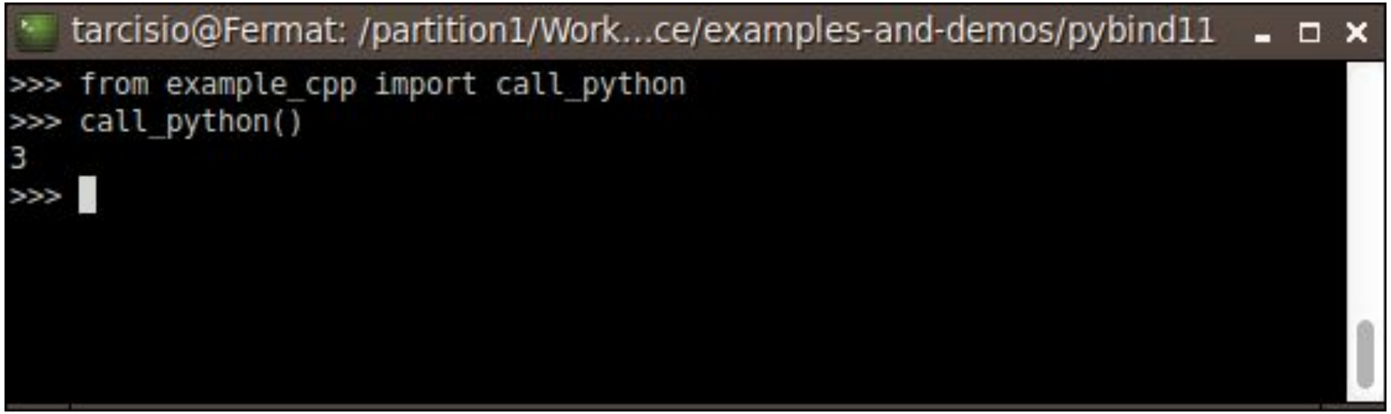
Import Python modules



```cpp
void call_python()
{
    py::object my_module = py::module::import("my_module");
    py::object result = my_module.attr("sum")(1, 2);
    py::print(result);
}
```

Import Python modules

```cpp
void call_python()
{
    py::object my_module = py::module::import("my_module");
    py::object result = my_module.attr("sum")(1, 2);
    py::print(result);
}
```



tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11

```
>>> from example_cpp import call_python
>>> call_python()
3
>>> 
```

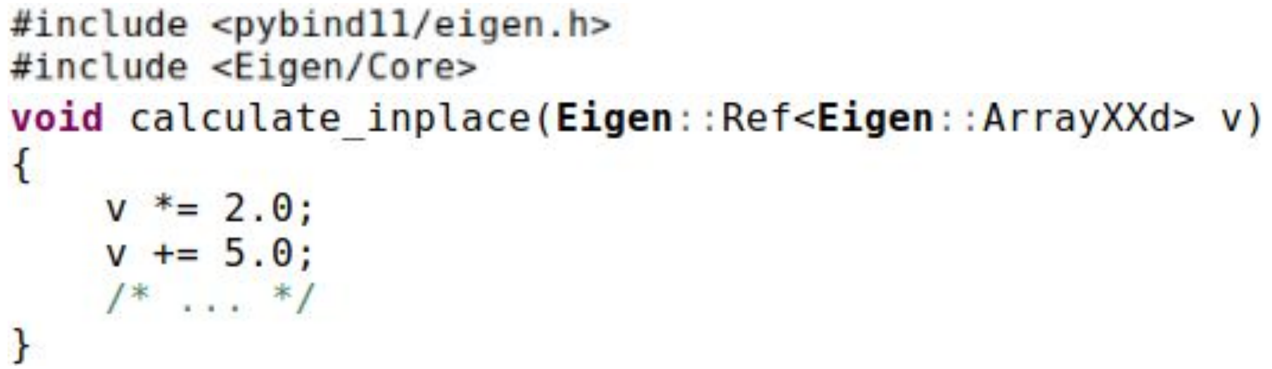
Numpy

Numpy

Basicamente três formas de interoperabilidade

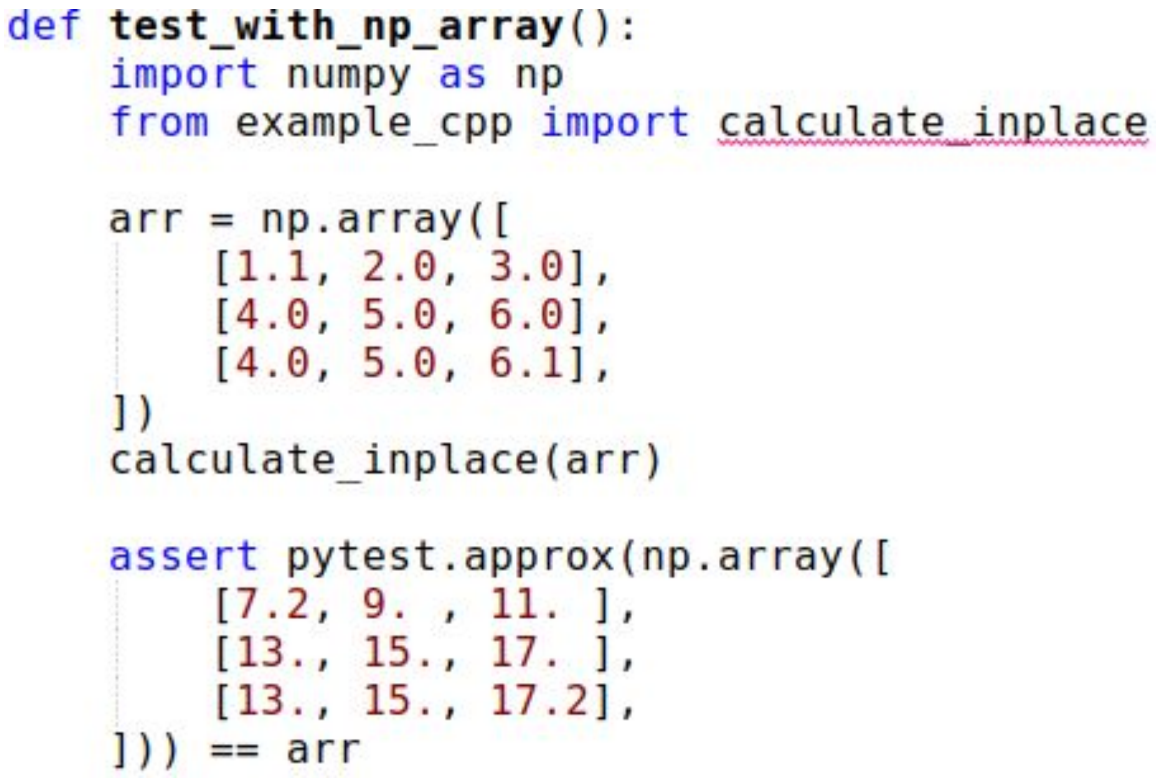
- **py::array / py::array_t<T>**
- **xTensor**
- **Eigen**

Numpy



```
#include <pybind11/eigen.h>
#include <Eigen/Core>
void calculate_inplace(Eigen::Ref<Eigen::ArrayXXd> v)
{
    v *= 2.0;
    v += 5.0;
    /* ... */
}
```

Numpy

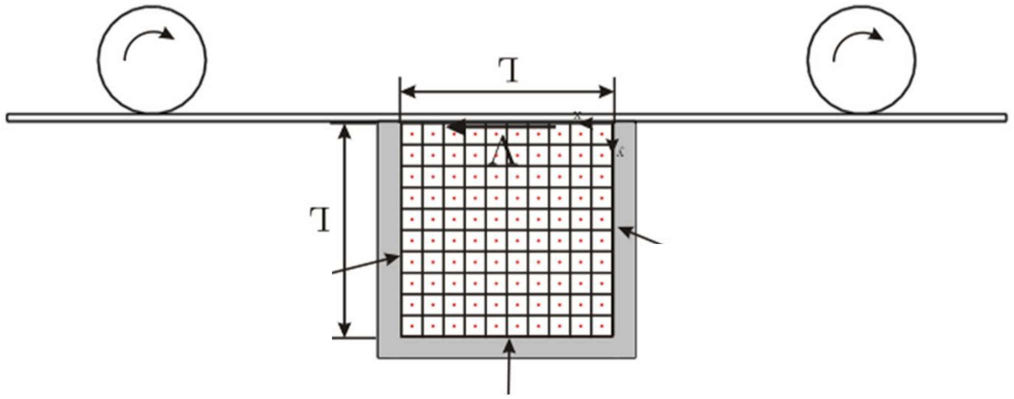


```python
def test_with_np_array():
    import numpy as np
    from example_cpp import calculate_inplace

    arr = np.array([
        [1.1, 2.0, 3.0],
        [4.0, 5.0, 6.0],
        [4.0, 5.0, 6.1],
    ])
    calculate_inplace(arr)

    assert pytest.approx(np.array([
        [7.2, 9.  , 11. ],
        [13., 15., 17. ],
        [13., 15., 17.2],
    ])) == arr
```

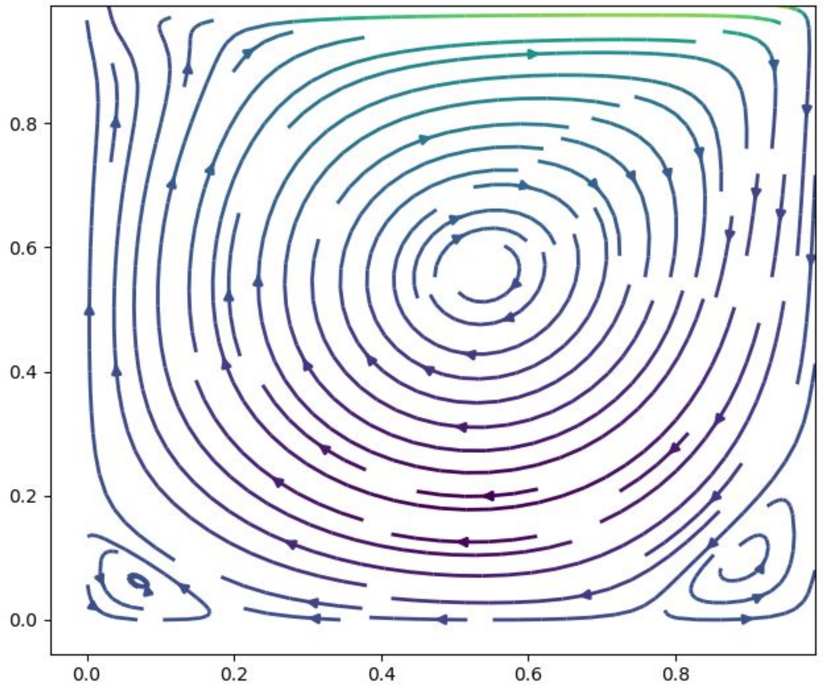
Básicos - Numpy

- Passagem transparente de vetores e matrizes para C++
- Retorno de numpy arrays para Python
- Passagem de matrizes por referência
- Escolha da ordenação das matrizes
  (Row Major vs Col Major)
- Proteção aos vetores ‘const’

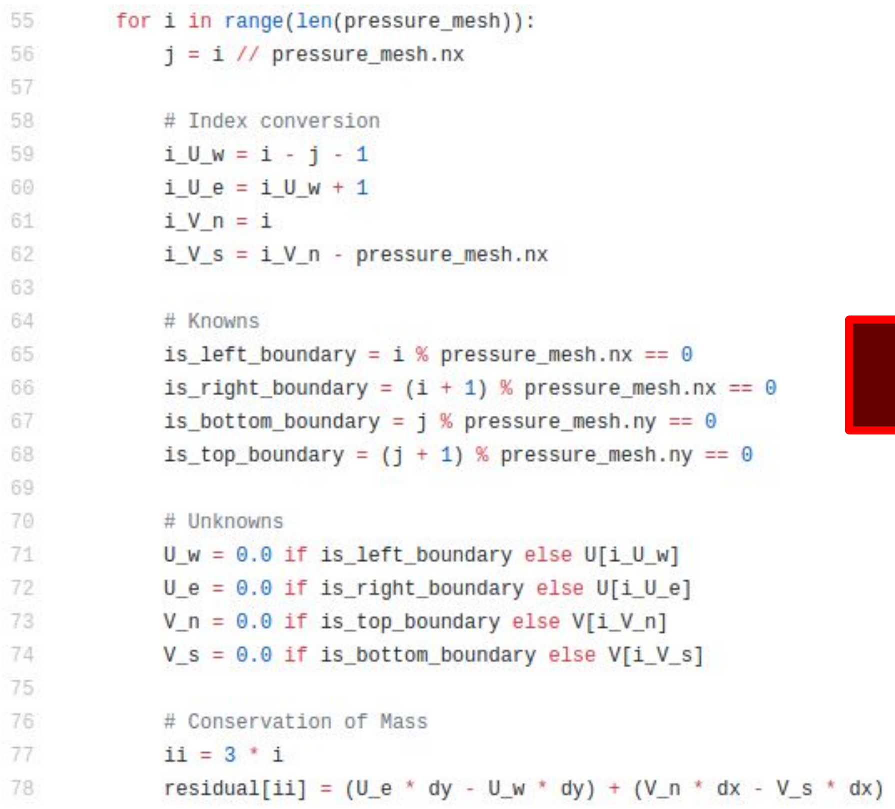
Exemplo de uso



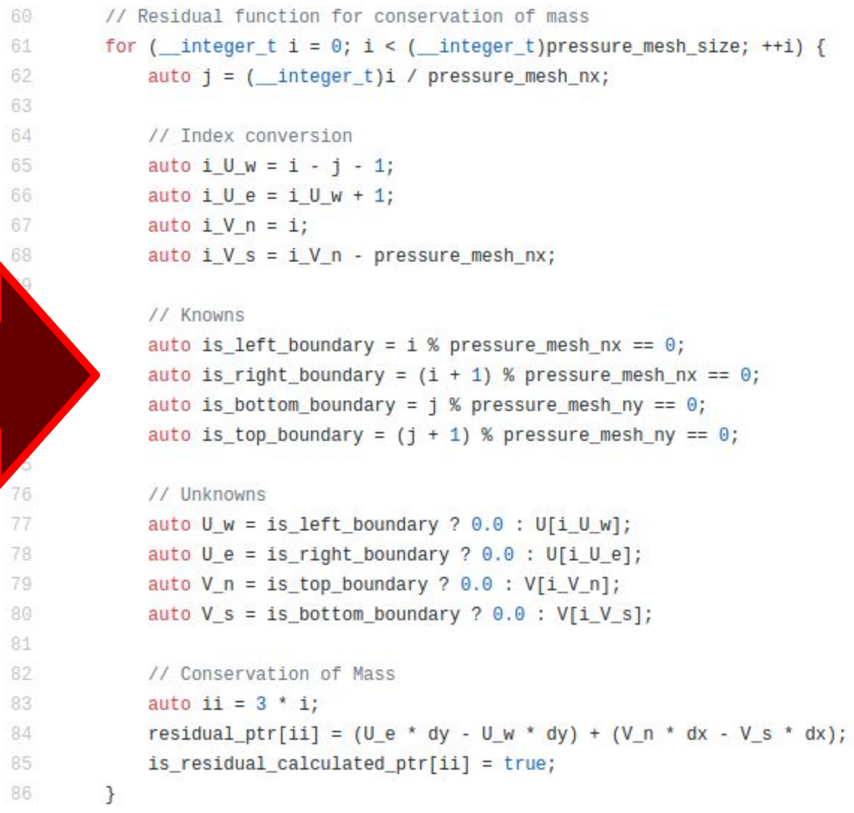
Exemplo de uso



Exemplo de uso

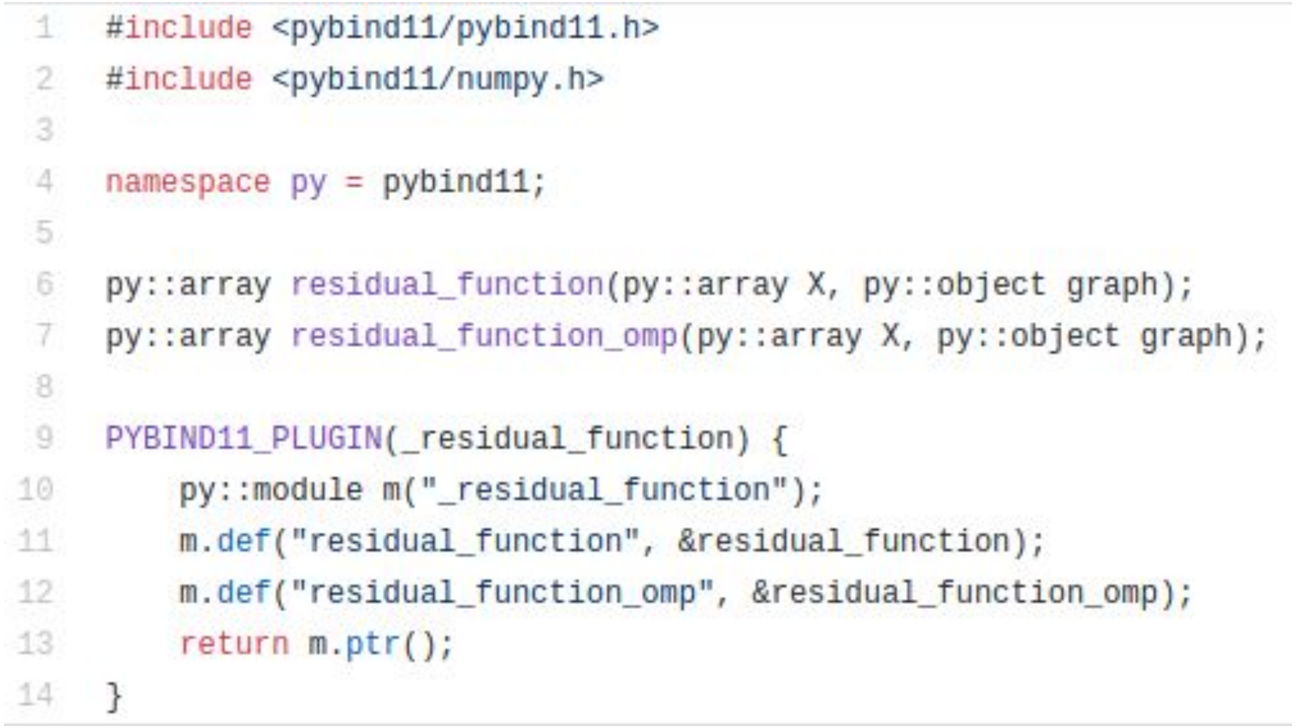


```python
for i in range(len(pressure_mesh)):
    j = i // pressure_mesh.nx

    # Index conversion
    i_U_w = i - j - 1
    i_U_e = i_U_w + 1
    i_V_n = i
    i_V_s = i_V_n - pressure_mesh.nx

    # Knowns
    is_left_boundary = i % pressure_mesh.nx == 0
    is_right_boundary = (i + 1) % pressure_mesh.nx == 0
    is_bottom_boundary = j % pressure_mesh.ny == 0
    is_top_boundary = (j + 1) % pressure_mesh.ny == 0

    # Unknowns
    U_w = 0.0 if is_left_boundary else U[i_U_w]
    U_e = 0.0 if is_right_boundary else U[i_U_e]
    V_n = 0.0 if is_top_boundary else V[i_V_n]
    V_s = 0.0 if is_bottom_boundary else V[i_V_s]

    # Conservation of Mass
    ii = 3 * i
    residual[ii] = (U_e * dy - U_w * dy) + (V_n * dx - V_s * dx)
```

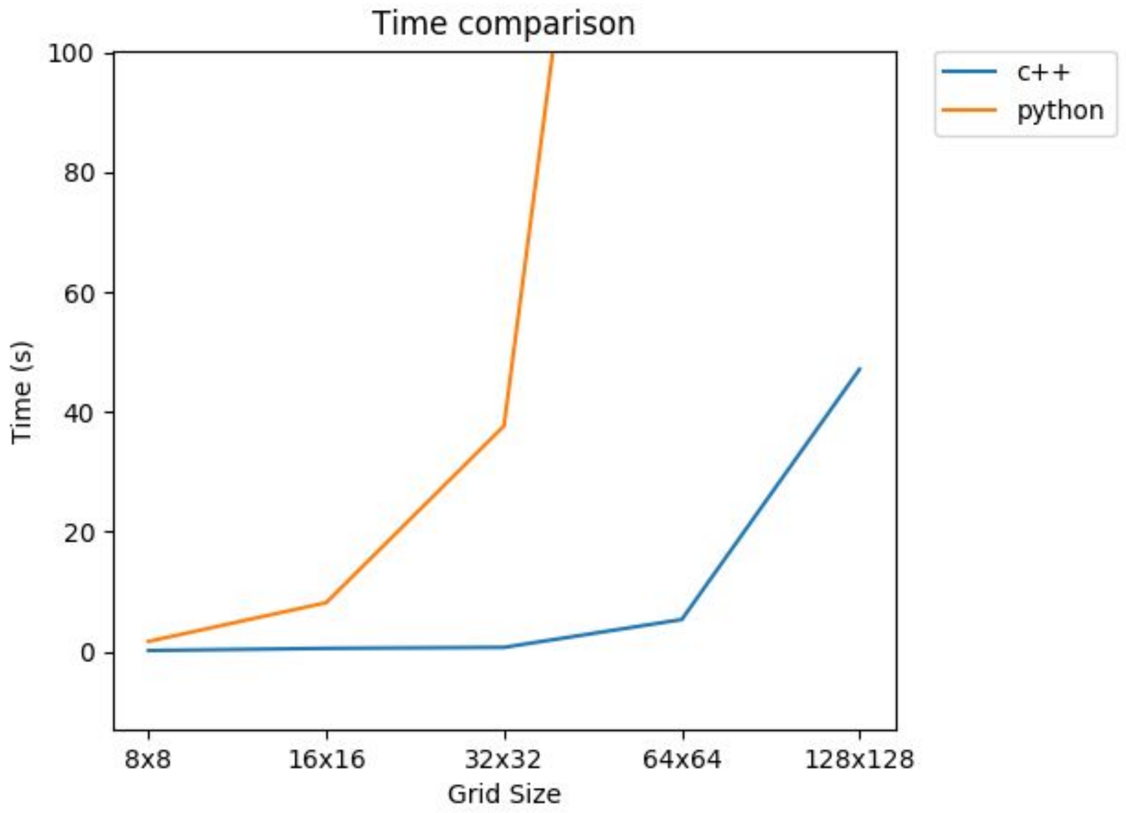


```cpp
// Residual function for conservation of mass
for (__integer_t i = 0; i < (__integer_t)pressure_mesh_size; ++i) {
    auto j = (__integer_t)i / pressure_mesh_nx;

    // Index conversion
    auto i_U_w = i - j - 1;
    auto i_U_e = i_U_w + 1;
    auto i_V_n = i;
    auto i_V_s = i_V_n - pressure_mesh_nx;

    // Knowns
    auto is_left_boundary = i % pressure_mesh_nx == 0;
    auto is_right_boundary = (i + 1) % pressure_mesh_nx == 0;
    auto is_bottom_boundary = j % pressure_mesh_ny == 0;
    auto is_top_boundary = (j + 1) % pressure_mesh_ny == 0;

    // Unknowns
    auto U_w = is_left_boundary ? 0.0 : U[i_U_w];
    auto U_e = is_right_boundary ? 0.0 : U[i_U_e];
    auto V_n = is_top_boundary ? 0.0 : V[i_V_n];
    auto V_s = is_bottom_boundary ? 0.0 : V[i_V_s];

    // Conservation of Mass
    auto ii = 3 * i;
    residual_ptr[ii] = (U_e * dy - U_w * dy) + (V_n * dx - V_s * dx);
    is_residual_calculated_ptr[ii] = true;
}
```

Exemplo de uso



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

namespace py = pybind11;

py::array residual_function(py::array X, py::object graph);
py::array residual_function_omp(py::array X, py::object graph);

PYBIND11_PLUGIN(_residual_function) {
    py::module m("_residual_function");
    m.def("residual_function", &residual_function);
    m.def("residual_function_omp", &residual_function_omp);
    return m.ptr();
}
```


Time comparison
Time (s)
Grid Size
100
80
60
40
20
0
8x8
16x16
32x32
64x64
128x128
c++
python

E o que mais?

- **GIL Release (Global Interpreter Lock)**
- **Exportar classes com herança e/ou herança multipla**
- **Exportar métodos virtuais (C++) para Python**
- **Exportar Enums**
- **Sobrecarga de operadores (Operator overloading)**
- **Suporte a smart pointers (std::unique_ptr, std::shared_ptr)**
- **Entre outras features...**

Obrigado