

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)

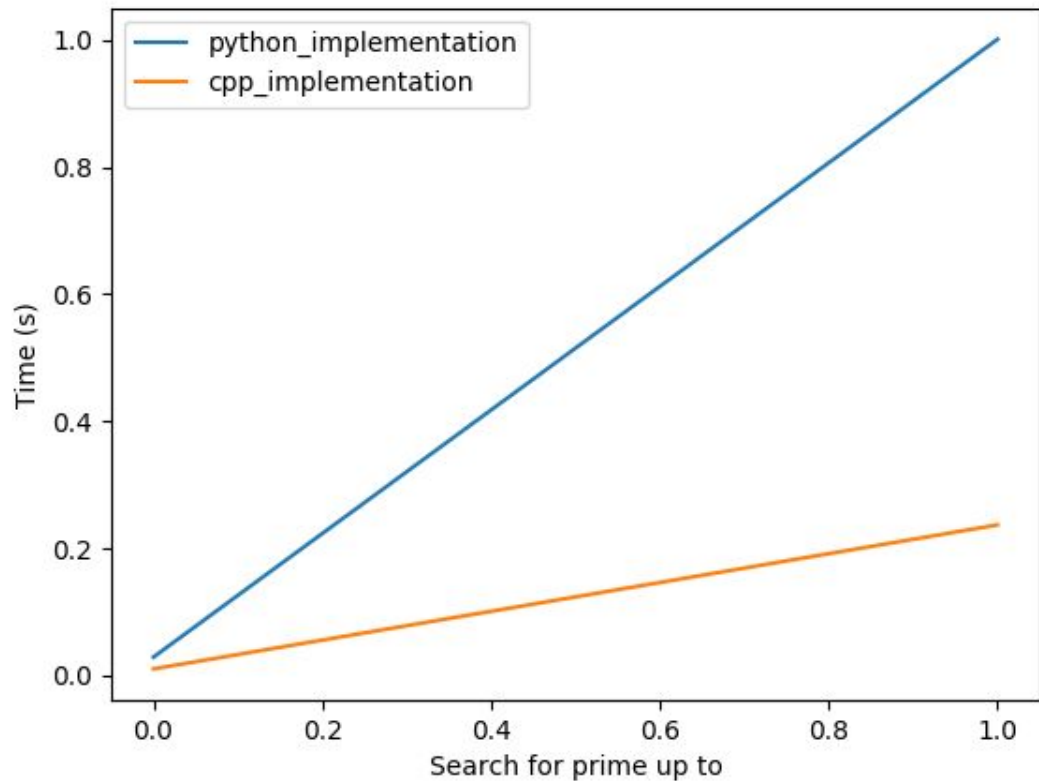
Motivação

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

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

```
3 bool is_prime(uint64_t n)
4 {
5     for (uint64_t k = 2; k < uint64_t(sqrt(n)) + 1; ++k) {
6         if (n % k == 0) {
7             return false;
8         }
9     }
10    return true;
11 }
```

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

```
1 #include <pybind11/pybind11.h>
2
3 int add(int i, int j) {
4     return i + j;
5 }
6
7 PYBIND11_MODULE(example_cpp, m) {
8     m.def("add", &add);
9 }
10
```

Básicos - Funções

```
1 #include <pybind11/pybind11.h>
2
3 int add(int i, int j) {
4     return i + j;
5 }
6
7 PYBIND11_MODULE(example_cpp, m) {
8     m.def("add", &add);
9 }
10
```

Básicos - Funções

```
1 #include <pybind11/pybind11.h>
2
3 int add(int i, int j) {
4     return i + j;
5 }
6
7 PYBIND11_MODULE(example_cpp, m) {
8     m.def("add", &add);
9 }
10
```


Básicos - Funções

```
1 #include <pybind11/pybind11.h>
2
3 int add(int i, int j) {
4     return i + j;
5 }
6
7 PYBIND11_MODULE(example_cpp, m) {
8     m.def("add", &add);
9 }
10
```

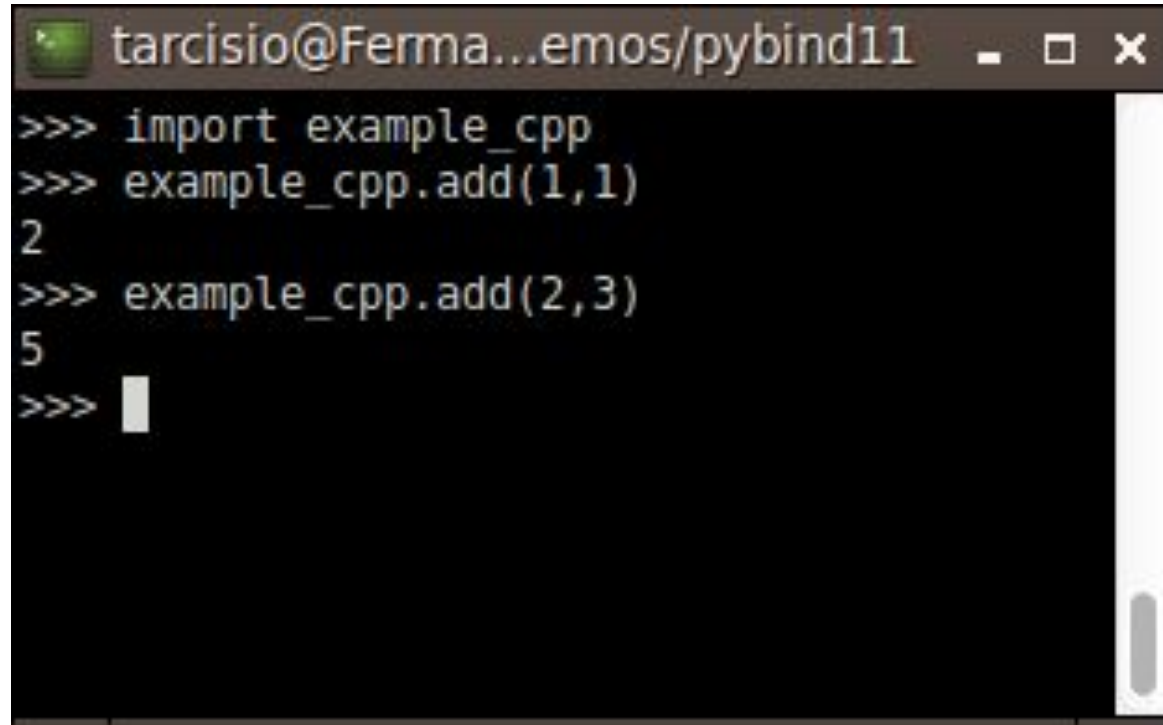
Básicos - Funções

```
1 #include <pybind11/pybind11.h>
2
3 int add(int i, int j) {
4     return i + j;
5 }
6
7 PYBIND11_MODULE(example_cpp, m) {
8     m.def("add", &add);
9 }
10
```

Básicos - Funções - Possível makefile (básico)*

```
1 all:
2     g++ \
3         -O2 \
4         -Wall \
5         -shared \
6         -std=c++11 \
7         -fPIC \
8         `python3 -m pybind11 --includes` \
9         example_cpp.cpp \
10        -o example_cpp.so
11
```

Básicos - Funções - Uso em python

A terminal window with a dark background and light-colored text. The window title bar shows the username 'tarcisio@Ferma...' and the directory path '...emos/pybind11'. The terminal content shows a Python interactive session where the 'example_cpp' module is imported, and its 'add' function is called twice with different arguments, returning the results 2 and 5 respectively. A cursor is visible on the line following the last command.

```
tarcisio@Ferma...emos/pybind11  
>>> import example_cpp  
>>> example_cpp.add(1,1)  
2  
>>> example_cpp.add(2,3)  
5  
>>> 
```

Básicos - Funções - Import error

```
tarcsio@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
```

```
1 all:
2     g++ \
3         -O2 \
4         -Wall \
5         -shared \
6         -std=c++11 \
7         -fPIC \
8         `python3 -m pybind11 --includes` \
9         example_cpp.cpp \
10        -o example_cpp.so
11
```

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'
```



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:

```
10 def find_next_prime(other_primes, start, stop, n_primes):
11     if n_primes <= 0:
12         return []
13     for i in range(start, stop):
14         if not is_prime(i):
15             continue
16         if len(other_primes) == 0:
17             try:
18                 return [i] + find_next_prime(
19                     [i],
20                     start + 1,
21                     stop,
22                     n_primes - 1
23                 )
24             except NextPrimeNotFound:
25                 continue
26     found = True
27     for j in other_primes:
28         if not is_prime(int("%s%s" % (j, i))):
29             found = False
30             break
31         if not is_prime(int("%s%s" % (i, j))):
32             found = False
33             break
```

```
34     if found:
35         if n_primes == 0:
36             return [i] + other_primes
37         else:
38             try:
39                 return [i] + find_next_prime(
40                     other_primes + [i],
41                     start + 1,
42                     stop,
43                     n_primes - 1
44                 )
45             except NextPrimeNotFound:
46                 continue
47     raise NextPrimeNotFound()
```

Exemplo 1:

```
10 def find_next_prime(other primes):
```

```
11     if n_primes <= 0:
```

```
12         return
```

```
13     for
```

```
14
```

```
15
```

```
16
```

```
17
```

```
18
```

```
19
```

```
20
```

```
21
```

```
22
```

```
23
```

```
24
```

```
25
```

```
26
```

```
27
```

```
28
```

```
29
```

```
30
```

```
31
```

```
32
```

```
33
```

```
    found
```

```
    for
```

```
    if
```

???

???

```
    0:
```

```
    + other_primes
```

```
    ] + find_next_prime(
```

```
    primes + [i],
```

```
    + 1,
```

```
    s - 1
```

```
    NotFound:
```

```
    raise NotFound()
```

Exemplo 1:

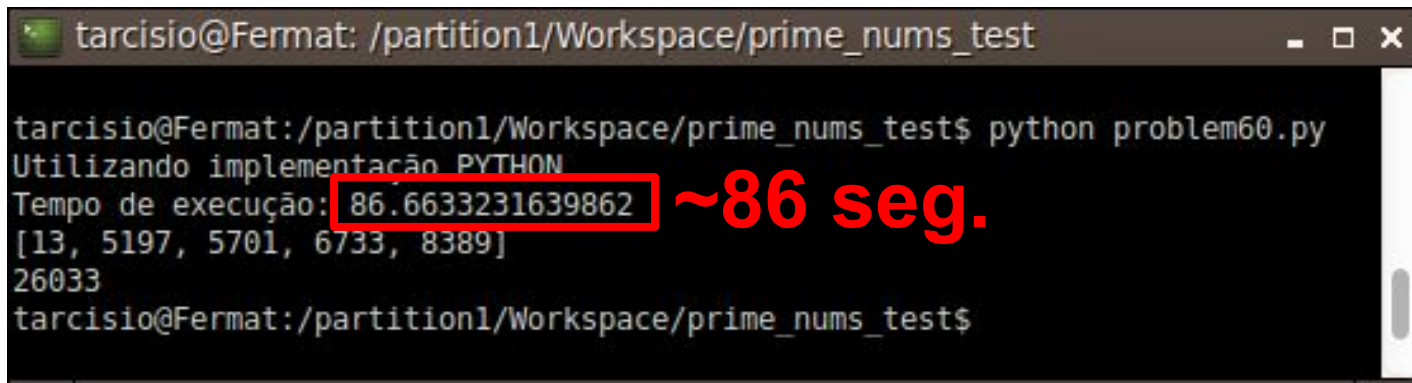
```
10 def find_next_prime(other_primes, start, stop, n_primes):
11     if n_primes <= 0:
12         return []
13     for i in range(start, stop):
14         if not is_prime(i):
15             continue
16         if len(other_primes) == 0:
17             try:
18                 return [i] + find_next_prime(
19                     [i],
20                     start + 1,
21                     stop,
22                     n_primes - 1
23                 )
24             except NextPrimeNotFound:
25                 continue
26         found = True
27         for j in other_primes:
28             if not is_prime(in ("%s%s" % (j, i))):
29                 found = False
30                 break
31             if not is_prime(in ("%s%s" % (i, j))):
32                 found = False
33                 break
```

```
34     if found:
35         if n_primes == 0:
36             return [i] + other_primes
37         else:
38             try:
39                 return [i] + find_next_prime(
40                     other_primes + [i],
41                     start + 1,
42                     stop,
43                     n_primes - 1
44                 )
45             except NextPrimeNotFound:
46                 continue
47     raise NextPrimeNotFound()
```

Exemplo 1:

```
1 #include <pybind11/pybind11.h>
2
3 bool is_prime(uint64_t n)
4 {
5     for (uint64_t k = 2; k < uint64_t(sqrt(n)) + 1; ++k) {
6         if (n % k == 0) {
7             return false;
8         }
9     }
10    return true;
11 }
12
13 PYBIND11_MODULE(cpp_implementation, m) {
14     m.def("is_prime", &is_prime);
15 }
16
```

Exemplo 1:



A terminal window titled "tarcisio@Fermat: /partition1/Workspace/prime_nums_test" with standard window controls. The terminal shows the execution of "python problem60.py". The output includes the text "Utilizando implementação PYTHON", the execution time "Tempo de execução: 86.6633231639862" (where the number is highlighted with a red box and followed by "~86 seg." in red), a list of numbers "[13, 5197, 5701, 6733, 8389]", and the number "26033". The prompt returns to "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 ~86 seg.
[13, 5197, 5701, 6733, 8389]
26033
tarcisio@Fermat: /partition1/Workspace/prime_nums_test$
```


Exemplo 1:

```
tarcisio@Fermat: /partition1/Workspace/prime_nums_test$  
  
tarcisio@Fermat: /partition1/Workspace/prime_nums_test$ pyth  
Utilizando implementação PYTHON  
Tempo de execução: 86.6633231639862 ~86 seg.  
[13, 5197, 5701, 6733, 8389]  
26033  
tarcisio@Fermat: /partition1/Workspace/prime_nums_test$
```

```
tarcisio@Fermat: /partition1/Workspace/prime_nums_test$  
  
tarcisio@Fermat: /partition1/Workspace/prime_nums_test$ pyth  
Utilizando implementação C++  
Tempo de execução: 16.627538204193115 ~16 seg.  
[13, 5197, 5701, 6733, 8389]  
26033  
tarcisio@Fermat: /partition1/Workspace/prime_nums_test$
```



BUT WAIT

Pybind11

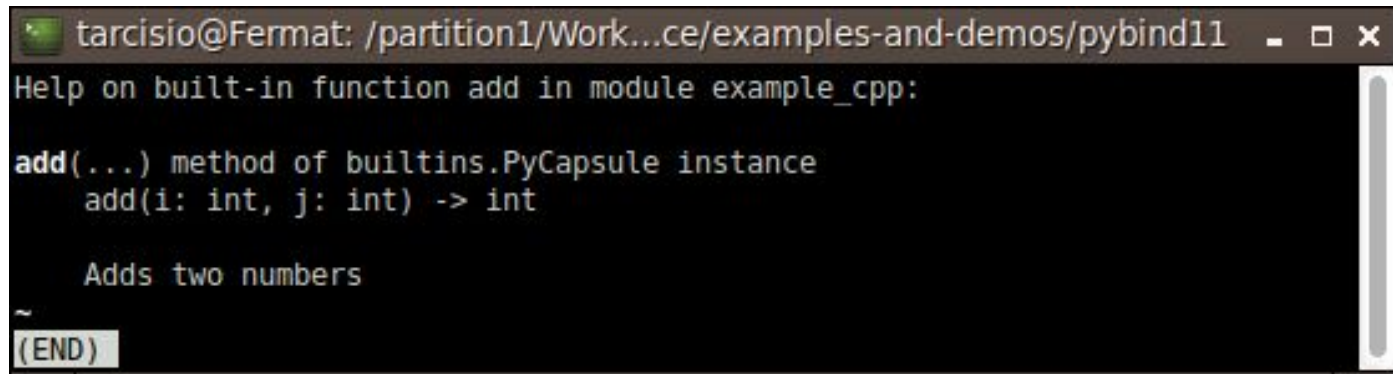
THERE'S MORE

Documentação & argumentos

```
95     m.def(  
96         "add",  
97         &add,  
98         "Adds two numbers",  
99         py::arg("i"),  
100        py::arg("j")  
101    );
```


Documentação & argumentos

```
95     m.def(  
96         "add",  
97         &add,  
98         "Adds two numbers",  
99         py::arg("i"),  
100        py::arg("j")  
101    );
```



A terminal window with a dark background and light text. The title bar shows the user 'tarcisio@Fermat' and the path '/partition1/Work...ce/examples-and-demos/pybind11'. The terminal content shows the help text for the 'add' function, including its signature and description. A vertical scrollbar is visible on the right side of the terminal window.

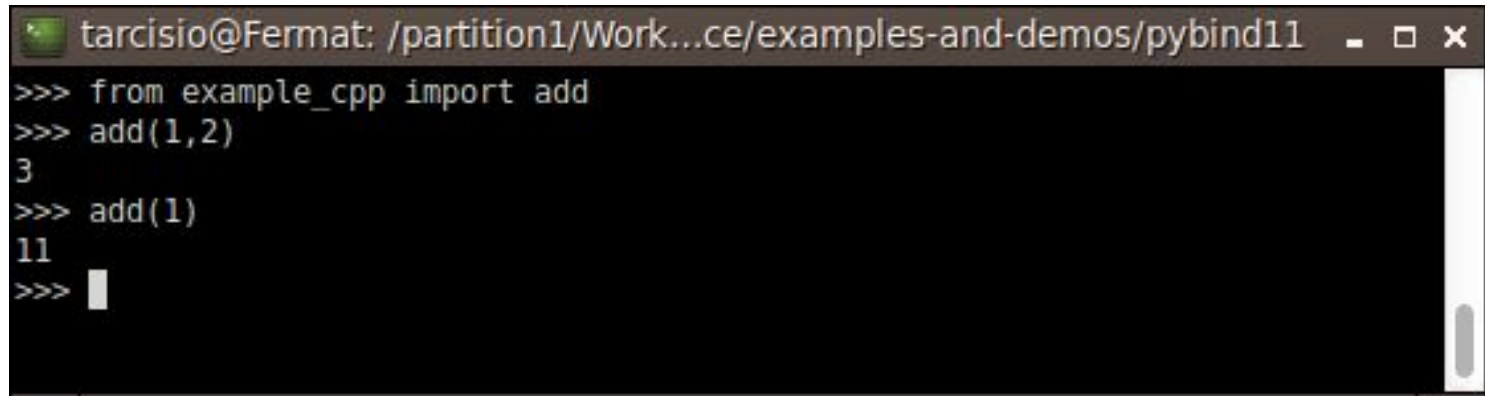
```
tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11  
Help on built-in function add in module example_cpp:  
  
add(...) method of builtins.PyCapsule instance  
    add(i: int, j: int) -> int  
  
    Adds two numbers  
~  
(END)
```

Parâmetros default

```
95     m.def(  
96         "add",  
97         &add,  
98         "Adds two numbers",  
99         py::arg("i"),  
100        py::arg("j")=10  
101    );
```

Parâmetros default

```
95     m.def(  
96         "add",  
97         &add,  
98         "Adds two numbers",  
99         py::arg("i"),  
100        py::arg("j")=10  
101    );
```



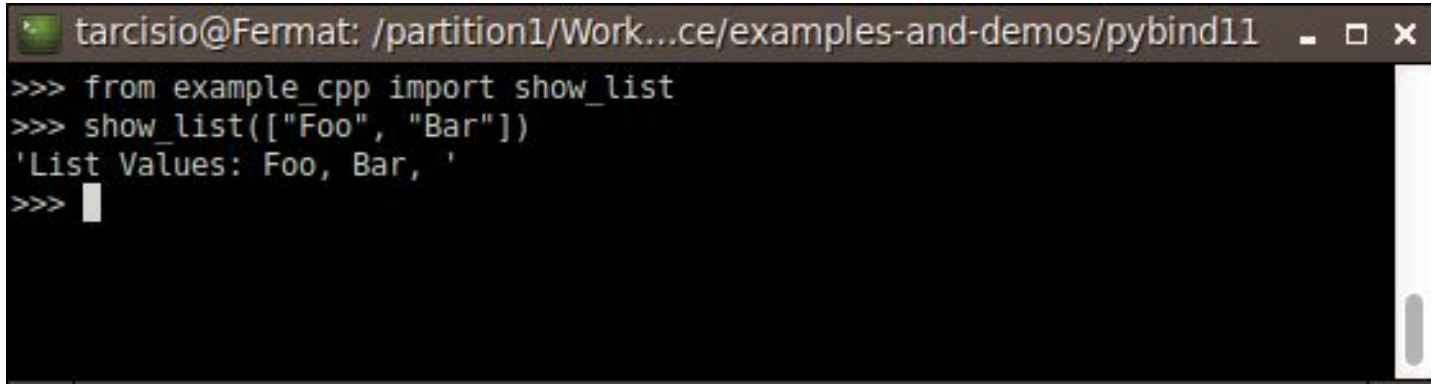
```
tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11  
>>> from example_cpp import add  
>>> add(1,2)  
3  
>>> add(1)  
11  
>>> 
```

Tipos Primitivos (C++)

```
4 std::string show_list(std::vector<std::string> const& values)
5 {
6     std::string r = "List Values: ";
7     for (auto v : values) {
8         r += v + ", ";
9     }
10    return r;
11 }
```

Tipos Primitivos (C++)

```
4 std::string show_list(std::vector<std::string> const& values)
5 {
6     std::string r = "List Values: ";
7     for (auto v : values) {
8         r += v + ", ";
9     }
10    return r;
11 }
```

A terminal window with a dark background. The title bar shows the user 'tarcisio@Fermat' and the current directory '/partition1/Work...ce/examples-and-demos/pybind11'. The terminal contains Python code that imports a C++ function 'show_list' from a module 'example_cpp'. The function is called with the arguments '["Foo", "Bar"]', and the output 'List Values: Foo, Bar, ' is displayed. The prompt '>>>' is followed by a cursor.

```
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++)

```
4 std::string show_list(std::vector<std::string> & values)
5 {
6     std::string r = "List Values: ";
7     for (auto v : values) {
8         r += v + ", ";
9     }
10
11     // Won't change python's object
12     values[0] = "Bar";
13
14     return r;
15 }
16
```

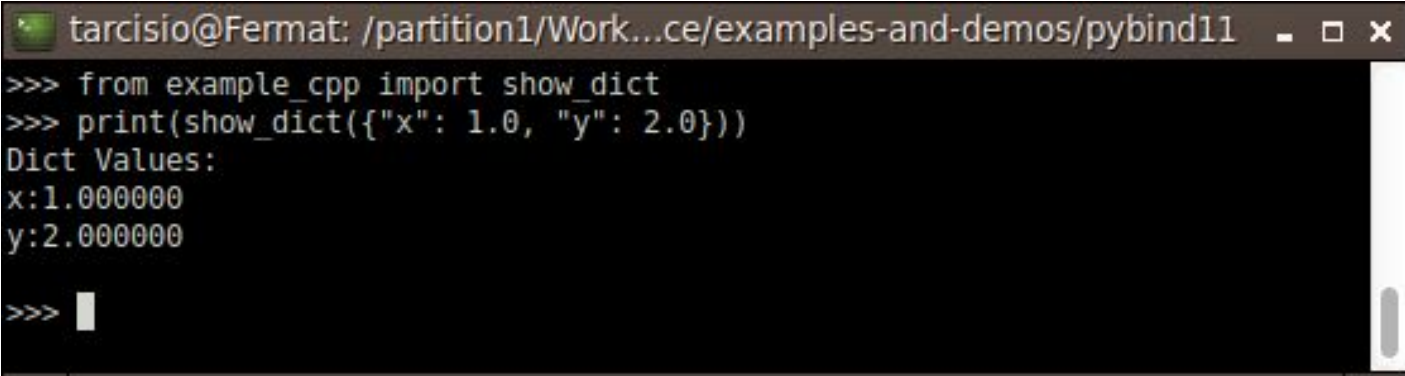
Apesar de aparecer como uma referência, o conversor do pybind11, na verdade, faz uma cópia de todos os dados antes de passá-los para uma função C++.

Tipos Primitivos (C++)

```
47 std::string show_dict(std::map<std::string, float> values)
48 {
49     std::string r = "Dict Values:\n";
50     for (auto kv : values) {
51         r += kv.first + ":" + std::to_string(kv.second) + "\n";
52     }
53
54     return r;
55 }
```

Tipos Primitivos (C++)

```
47 std::string show_dict(std::map<std::string, float> values)
48 {
49     std::string r = "Dict Values:\n";
50     for (auto kv : values) {
51         r += kv.first + ":" + std::to_string(kv.second) + "\n";
52     }
53
54     return r;
55 }
```

A terminal window with a dark background and light-colored text. The title bar shows the user 'tarcisio@Fermat' and the current directory '/partition1/Work...ce/examples-and-demos/pybind11'. The terminal contains Python code that imports a C++ function 'show_dict' from a module 'example_cpp'. It then calls 'show_dict' with a dictionary containing 'x': 1.0 and 'y': 2.0. The output of the function is printed, showing 'Dict Values:' followed by 'x:1.000000' and 'y:2.000000' on separate lines. A prompt '>>>>' is visible at the bottom left.

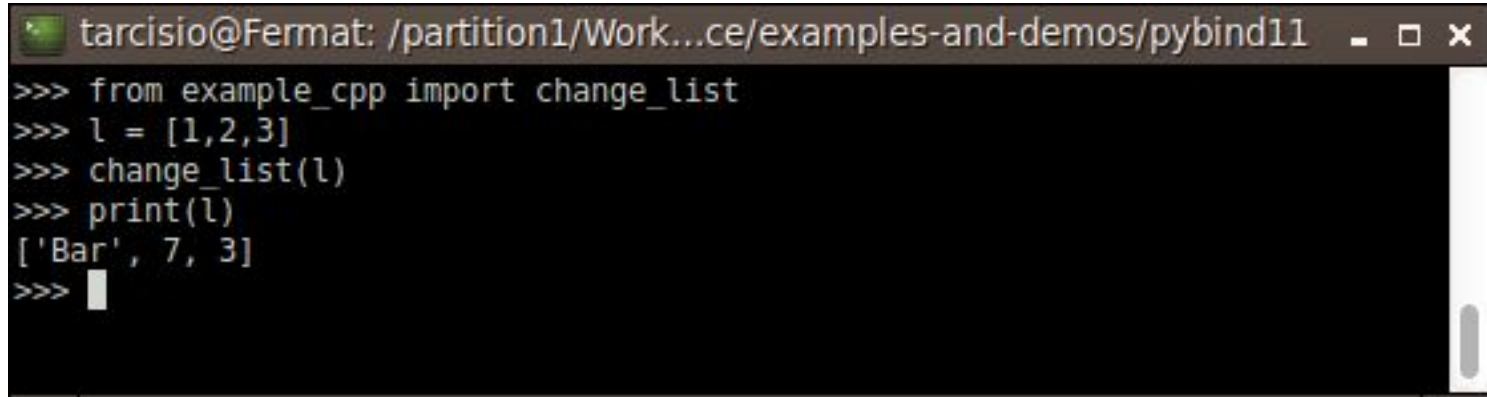
```
tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11
>>> from example_cpp import show_dict
>>> print(show_dict({"x": 1.0, "y": 2.0}))
Dict Values:
x:1.000000
y:2.000000
>>> 
```


Tipos Primitivos (Python)

```
6 void change_list(py::list values)
7 {
8     values[0] = "Bar";
9     values[1] = 7;
10 }
11
```

Tipos Primitivos (Python)

```
6 void change_list(py::list values)
7 {
8     values[0] = "Bar";
9     values[1] = 7;
10 }
11
```

A terminal window with a dark background and light text. The title bar shows the user 'tarcisio@Fermat' and the path '/partition1/Work...ce/examples-and-demos/pybind11'. The terminal contains a Python REPL session where the 'change_list' function from 'example_cpp' is imported. A list 'l' is created with values [1, 2, 3], and 'change_list(l)' is called. The output shows the list has been modified to ['Bar', 7, 3].

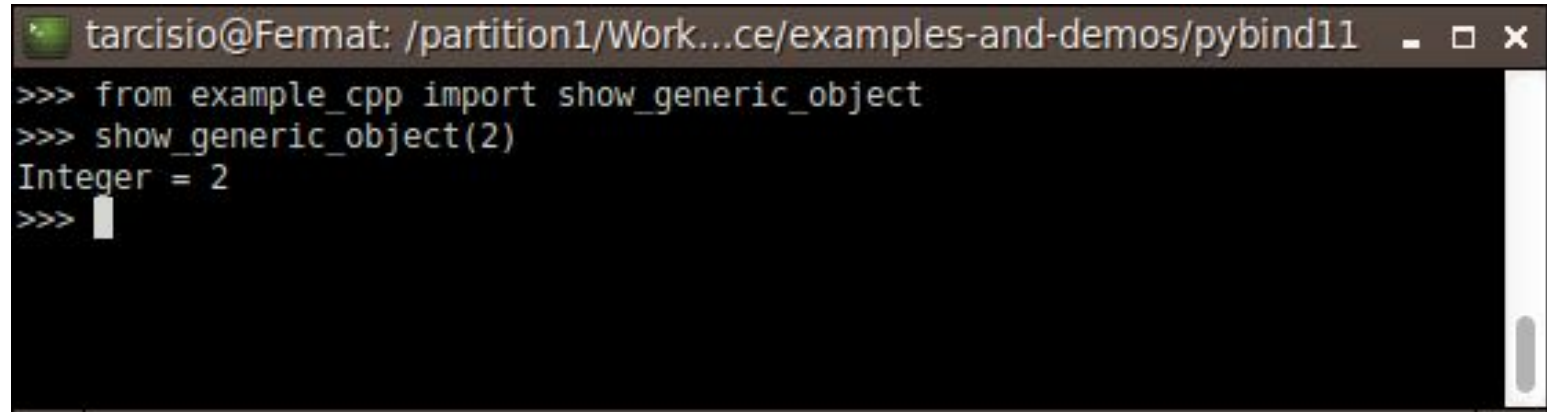
```
tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11
>>> from example_cpp import change_list
>>> l = [1,2,3]
>>> change_list(l)
>>> print(l)
['Bar', 7, 3]
>>>
```

Tipos Primitivos (Python)

```
35 void show_generic_object(py::object pyobj)
36 {
37     int i = pyobj.cast<int>();
38     std::cout << "Integer = " << std::to_string(i) << "\n";
39 }
```

Tipos Primitivos (Python)

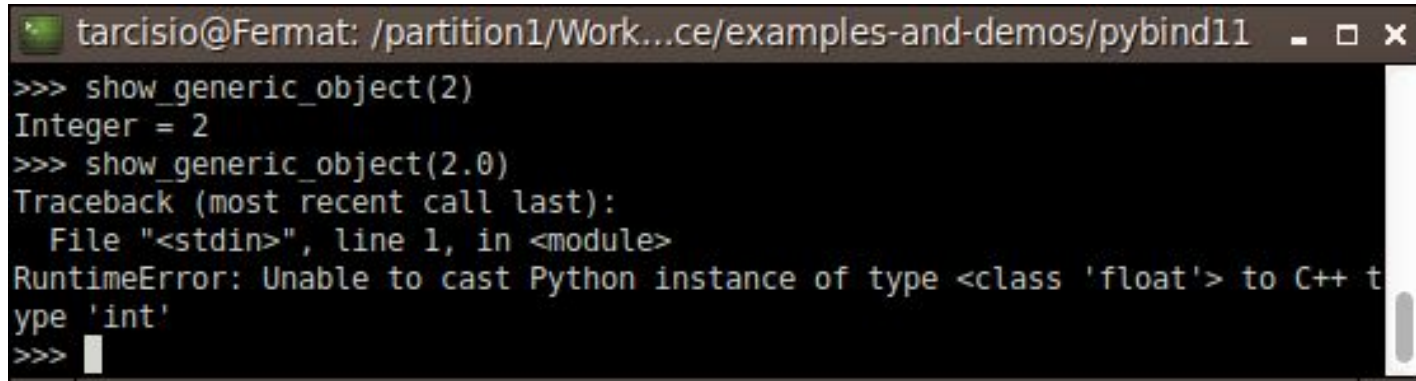
```
35 void show_generic_object(py::object pyobj)
36 {
37     int i = pyobj.cast<int>();
38     std::cout << "Integer = " << std::to_string(i) << "\n";
39 }
```

A terminal window with a dark background and light-colored text. The title bar shows the user 'tarcisio@Fermat' and the path '/partition1/Work...ce/examples-and-demos/pybind11'. The terminal contains the following text:

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

Conversão automática de exceções

```
35 void show_generic_object(py::object pyobj)
36 {
37     int i = pyobj.cast<int>();
38     std::cout << "Integer = " << std::to_string(i) << "\n";
39 }
```



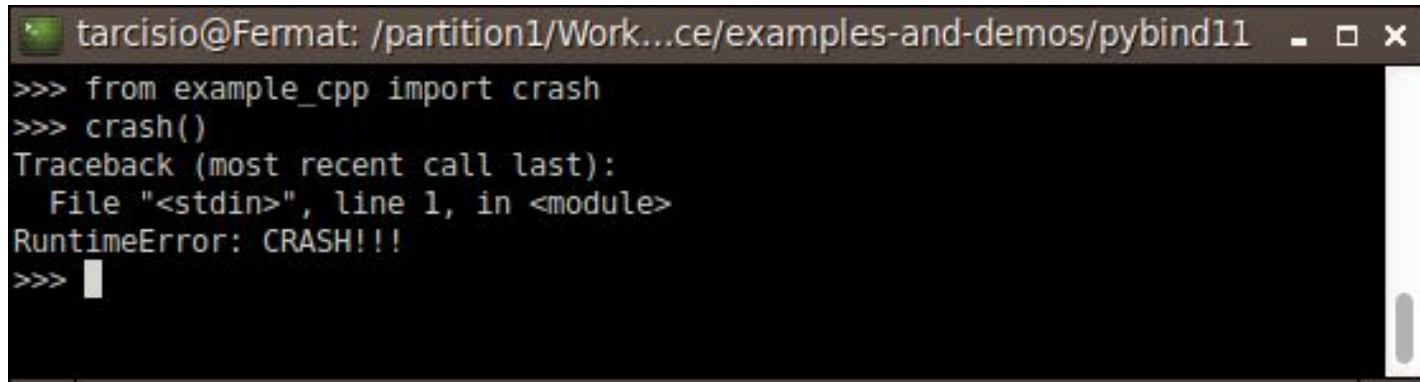
```
tarcísio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11
>>> show_generic_object(2)
Integer = 2
>>> show_generic_object(2.0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: Unable to cast Python instance of type <class 'float'> to C++ type 'int'
>>>
```

Conversão automática de exceções

```
35 void crash()  
36 {  
37     throw std::runtime_error("CRASH!!!");  
38 }
```

Conversão automática de exceções

```
35 void crash()  
36 {  
37     throw std::runtime_error("CRASH!!!");  
38 }
```



A terminal window titled "tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11" showing a Python REPL session. The user imports a C++ module and calls a function that throws a C++ exception. The exception is caught and converted to a Python RuntimeError.

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

Conversão automática de exceções

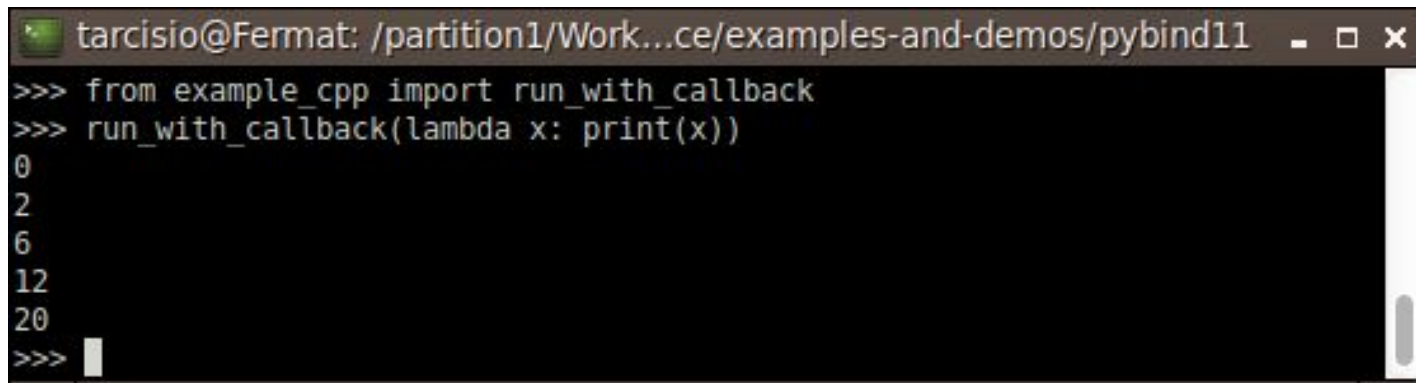
C++ exception type	Python exception type
<code>std::exception</code>	<code>RuntimeError</code>
<code>std::bad_alloc</code>	<code>MemoryError</code>
<code>std::domain_error</code>	<code>ValueError</code>
<code>std::invalid_argument</code>	<code>ValueError</code>
<code>std::length_error</code>	<code>ValueError</code>
<code>std::out_of_range</code>	<code>ValueError</code>
<code>std::range_error</code>	<code>ValueError</code>

Funções como argumento

```
105 void run_with_callback(std::function<void(int)> callback)
106 {
107     auto sum = 0;
108     for (auto i = 0; i < 5; ++i) {
109         sum += 2 * i;
110         callback(sum);
111     }
112 }
```

Funções como argumento

```
105 void run_with_callback(std::function<void(int)> callback)
106 {
107     auto sum = 0;
108     for (auto i = 0; i < 5; ++i) {
109         sum += 2 * i;
110         callback(sum);
111     }
112 }
```



A terminal window titled "tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11" showing the execution of the C++ code. The user enters two commands: `from example_cpp import run_with_callback` and `run_with_callback(lambda x: print(x))`. The output shows the sequence of values printed by the callback function: 0, 2, 6, 12, and 20, each on a new line. The prompt `>>>` is visible at the bottom.

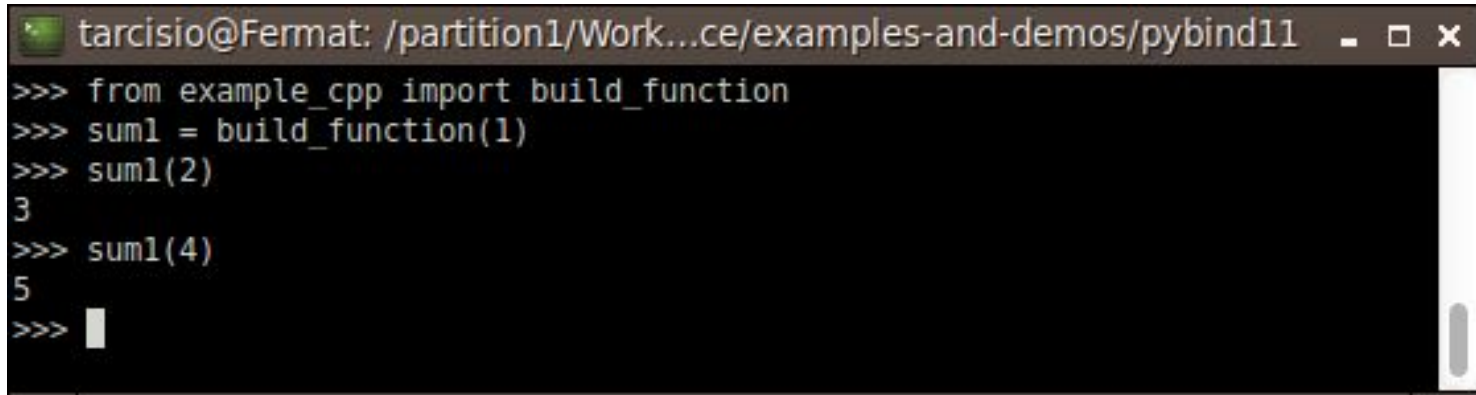
```
tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11
>>> from example_cpp import run_with_callback
>>> run_with_callback(lambda x: print(x))
0
2
6
12
20
>>>
```

Funções como argumento

```
114 std::function<int(int)> build_function(int i)
115 {
116     return [i](int j) {
117         return i + j;
118     };
119 }
```

Funções como argumento

```
114 std::function<int(int)> build_function(int i)
115 {
116     return [i](int j) {
117         return i + j;
118     };
119 }
```

A terminal window with a dark background and light text. The title bar shows the user 'tarcisio@Fermat' and the directory path '/partition1/Work...ce/examples-and-demos/pybind11'. The terminal contains a Python REPL session where the 'build_function' from 'example_cpp' is imported. Then, 'sum1' is created by calling 'build_function(1)'. Subsequent calls to 'sum1(2)' and 'sum1(4)' return the values 3 and 5 respectively. A cursor is visible at the end of the last prompt line.

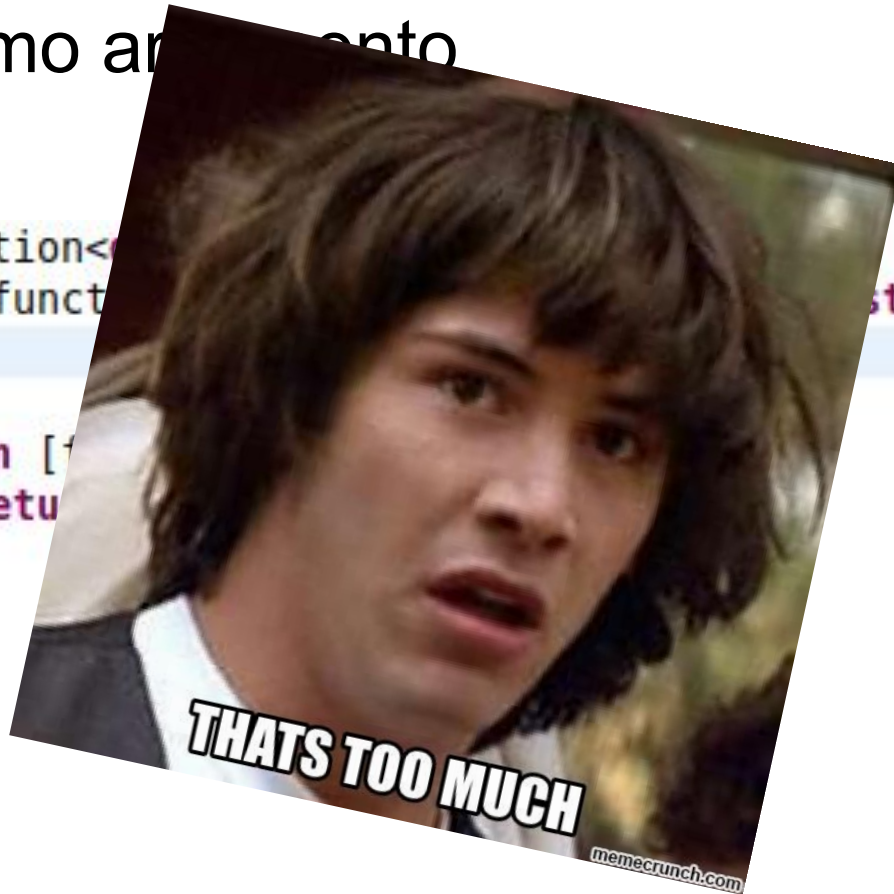
```
tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11
>>> from example_cpp import build_function
>>> sum1 = build_function(1)
>>> sum1(2)
3
>>> sum1(4)
5
>>> 
```

Funções como argumento

```
12 std::function<double(double)> build_function2(  
13     std::function<double(double, double)> const& f, double y  
14 )  
15 {  
16     return [f, y](double x) -> double {  
17         return f(x, y);  
18     };  
19 }
```

Funções como argumento

```
12 std::function<  
13     std::function<double (const T& f, double y  
14 )  
15 {  
16     return [f]  
17     {  
18         return f(y);  
19     };  
19 }
```



Classes & Objetos

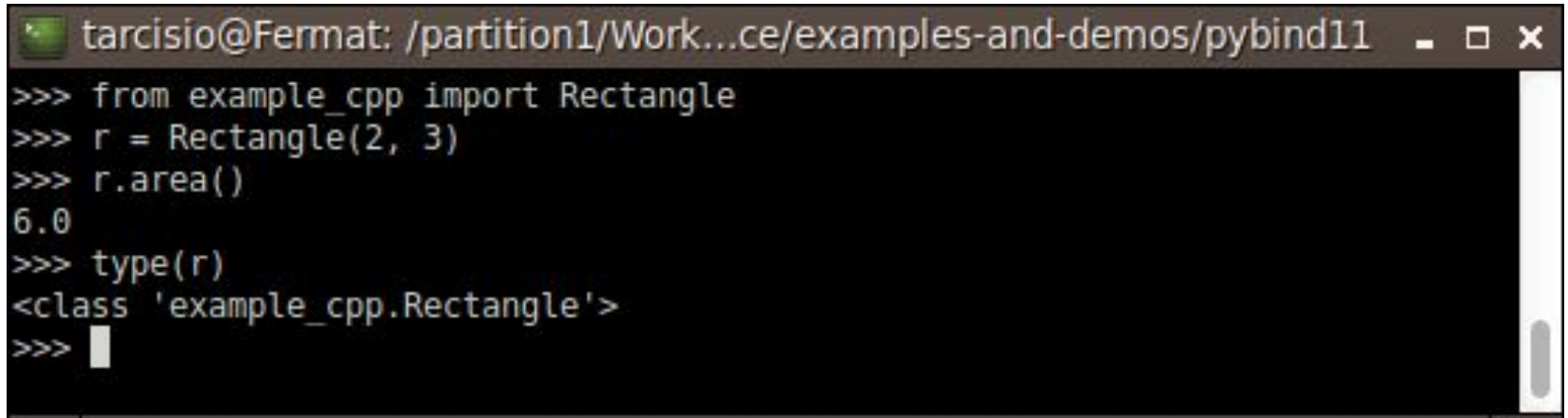
```
128 class Rectangle : public GeometricObject2d {
129 public:
130     Rectangle(double width, double height)
131         : _width(width), _height(height) {}
132
133     double area() override const { return this->_width * this->_height; }
134
135     double getWidth() const { return this->_width; }
136     void setWidth(double w) { this->_width = w; }
137 private:
138     double _width;
139     double _height;
140 };
```

Classes & Objetos

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


Classes & Objetos

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

A terminal window with a dark background and light-colored text. The title bar shows the user 'tarcisio@Fermat' and the directory path '/partition1/Work...ce/examples-and-demos/pybind11'. The terminal contains a Python REPL session where a C++ module 'example_cpp' is imported, a 'Rectangle' object is created with parameters (2, 3), its 'area' method is called, and its type is printed. The output shows the area is 6.0 and the type is '<class 'example_cpp.Rectangle'>'.

```
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

```
5 from example_cpp import Rectangle
6 class Square(Rectangle):
7     def __init__(self, side):
8         Rectangle.__init__(self, side, side)
9
```

Classes & Objetos

```
5 from example_cpp import Rectangle
6 class Square(Rectangle):
7     def __init__(self, side):
8         Rectangle.__init__(self, side, side)
9
```

```
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

```
163     py::class_<Rectangle>(m, "Rectangle")
164         .def(py::init<double, double>())
165         .def("area", &Rectangle::area)
166         .def_property(
167             "width",
168             &Rectangle::getWidth,
169             &Rectangle::setWidth
170         );
```

Classes & Objetos

```
163     py::class_<Rectangle>(m, "Rectangle")
164         .def(py::init<double, double>())
165         .def("area", &Rectangle::area)
166         .def_property(
167             "width",
168             &Rectangle::getWidth,
169             &Rectangle::setWidth
170         );
```

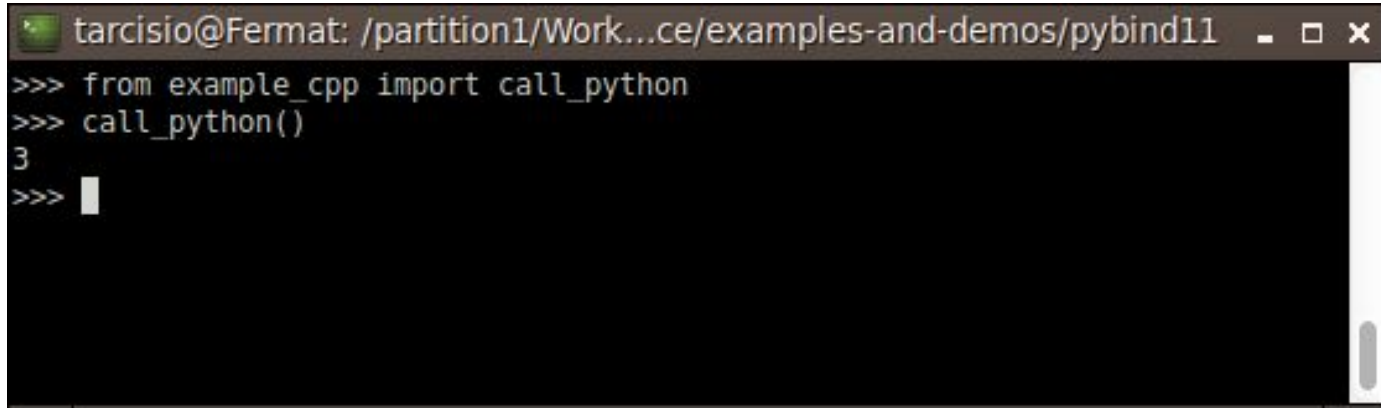
```
tarcisio@Fermat: /partition1/Work...ce/examples-and-demos/pybind11
>>> from example_cpp import Rectangle
>>> r = Rectangle(2, 3)
>>> r.width
2.0
>>> r.height
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'example_cpp.Rectangle' object has no attribute 'height'
>>>
```

Import Python modules

```
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

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

A terminal window with a dark background and light text. The title bar shows the user 'tarcisio@Fermat' and the current directory '/partition1/Work...ce/examples-and-demos/pybind11'. The terminal content shows a Python prompt '>>>' followed by the command 'from example_cpp import call_python', another prompt '>>>' followed by 'call_python()', and the output '3'. A third prompt '>>>' is followed by a cursor.

```
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

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

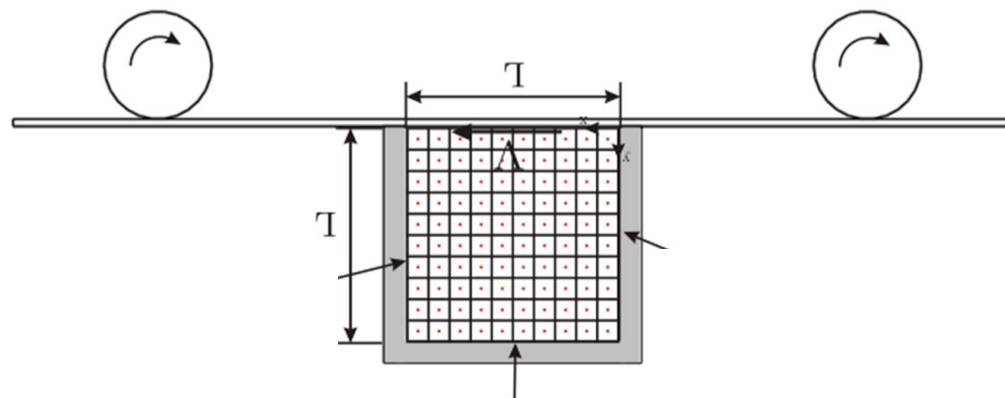
    arr = np.array([
        [1.1, 2.0, 3.0],
        [4.0, 5.0, 6.0],
        [4.0, 5.0, 6.1],
    ])
    calculate_inplace(arr)

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

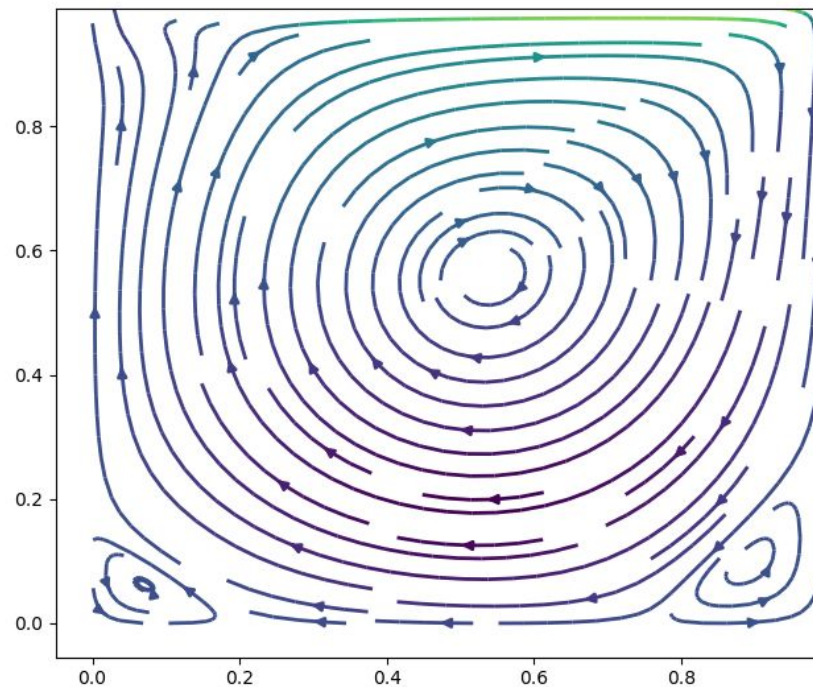
Básicos - Numpy

- **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

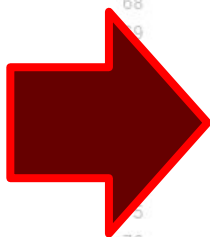


Exemplo de uso



Exemplo de uso

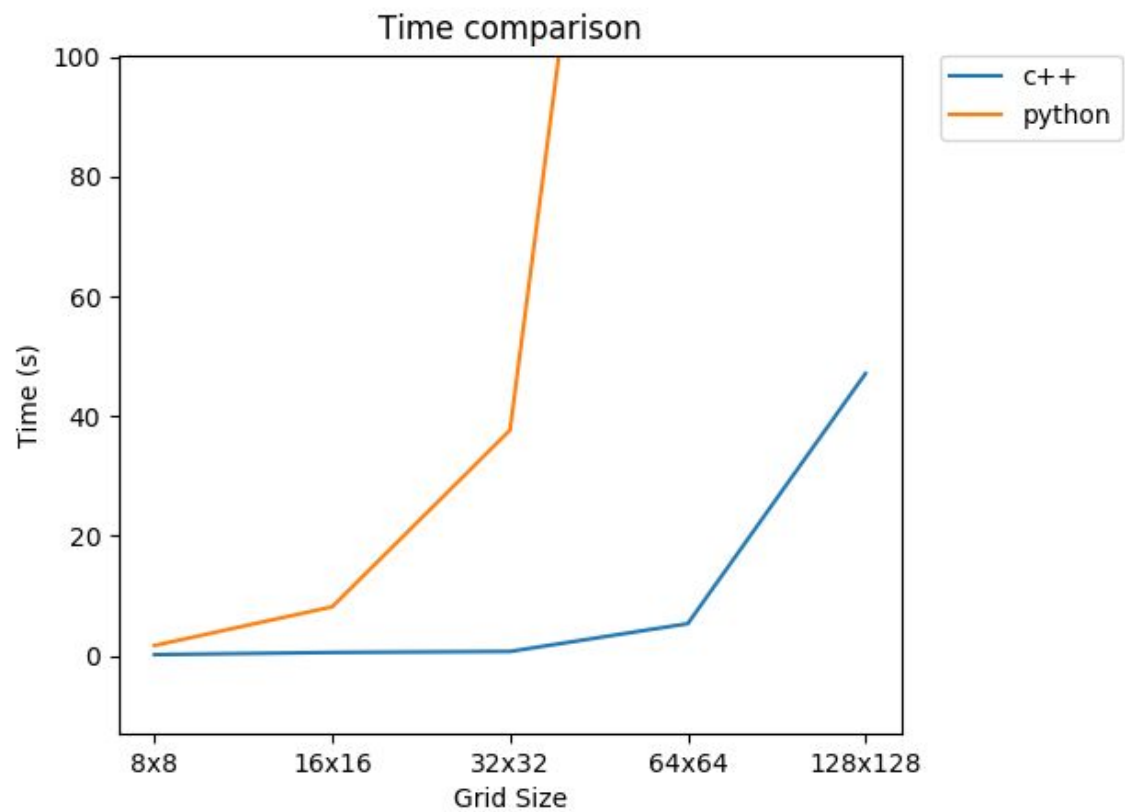
```
55     for i in range(len(pressure_mesh)):
56         j = i // pressure_mesh.nx
57
58         # Index conversion
59         i_U_w = i - j - 1
60         i_U_e = i_U_w + 1
61         i_V_n = i
62         i_V_s = i_V_n - pressure_mesh.nx
63
64         # Knowns
65         is_left_boundary = i % pressure_mesh.nx == 0
66         is_right_boundary = (i + 1) % pressure_mesh.nx == 0
67         is_bottom_boundary = j % pressure_mesh.ny == 0
68         is_top_boundary = (j + 1) % pressure_mesh.ny == 0
69
70         # Unknowns
71         U_w = 0.0 if is_left_boundary else U[i_U_w]
72         U_e = 0.0 if is_right_boundary else U[i_U_e]
73         V_n = 0.0 if is_top_boundary else V[i_V_n]
74         V_s = 0.0 if is_bottom_boundary else V[i_V_s]
75
76         # Conservation of Mass
77         ii = 3 * i
78         residual[ii] = (U_e * dy - U_w * dy) + (V_n * dx - V_s * dx)
```



```
60     // Residual function for conservation of mass
61     for (__integer_t i = 0; i < (__integer_t)pressure_mesh_size; ++i) {
62         auto j = (__integer_t)i / pressure_mesh_nx;
63
64         // Index conversion
65         auto i_U_w = i - j - 1;
66         auto i_U_e = i_U_w + 1;
67         auto i_V_n = i;
68         auto i_V_s = i_V_n - pressure_mesh_nx;
69
70         // Knowns
71         auto is_left_boundary = i % pressure_mesh_nx == 0;
72         auto is_right_boundary = (i + 1) % pressure_mesh_nx == 0;
73         auto is_bottom_boundary = j % pressure_mesh_ny == 0;
74         auto is_top_boundary = (j + 1) % pressure_mesh_ny == 0;
75
76         // Unknowns
77         auto U_w = is_left_boundary ? 0.0 : U[i_U_w];
78         auto U_e = is_right_boundary ? 0.0 : U[i_U_e];
79         auto V_n = is_top_boundary ? 0.0 : V[i_V_n];
80         auto V_s = is_bottom_boundary ? 0.0 : V[i_V_s];
81
82         // Conservation of Mass
83         auto ii = 3 * i;
84         residual_ptr[ii] = (U_e * dy - U_w * dy) + (V_n * dx - V_s * dx);
85         is_residual_calculated_ptr[ii] = true;
86     }
```

Exemplo de uso

```
1  #include <pybind11/pybind11.h>
2  #include <pybind11/numpy.h>
3
4  namespace py = pybind11;
5
6  py::array residual_function(py::array X, py::object graph);
7  py::array residual_function_omp(py::array X, py::object graph);
8
9  PYBIND11_PLUGIN(_residual_function) {
10     py::module m("_residual_function");
11     m.def("residual_function", &residual_function);
12     m.def("residual_function_omp", &residual_function_omp);
13     return m.ptr();
14 }
```

E o que mais?

- **GIL Release (Global Interpreter Lock)**
- **Exportar classes com herança e/ou herança múltipla**
- **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