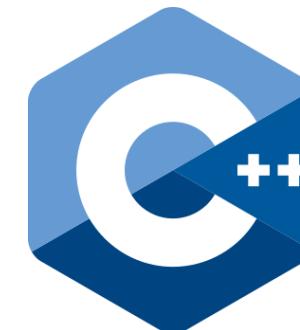
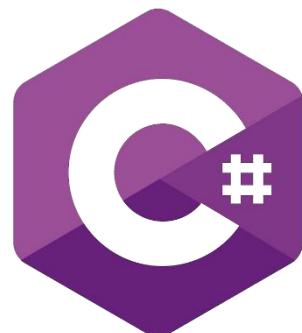


Interfaçage du C#, Python, C, C++, Go et Rust avec LabVIEW



Sommaire

I. Interfaçage de fonction

1. C
2. C++
3. Python
4. Go
5. Rust

II. Interfaçage d'objet

1. Objet
2. C#
3. Python
4. C++ vers Python (avec SWIG ou PyBind11)
5. C++

III. Conclusion

IV. Reference

V. Annexe

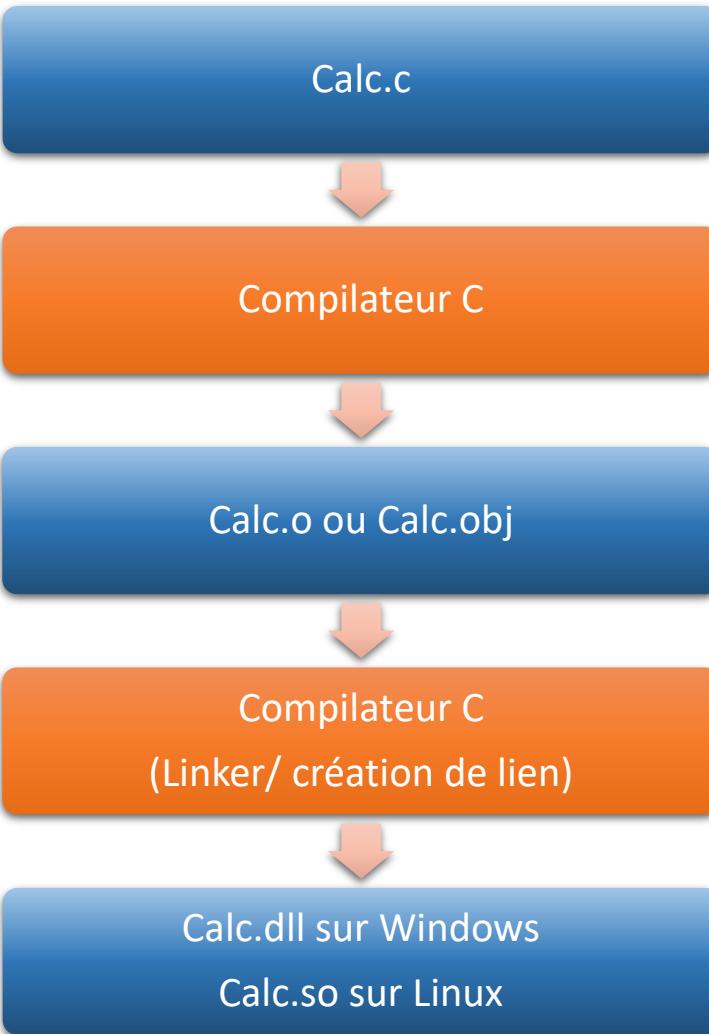
I. Interfaçage de fonction

1. C
2. C++
3. Python
4. Go
5. Rust

Des fonctions C dans LabVIEW



Fonctions C (1/2)



The terminal window shows the following commands being run:

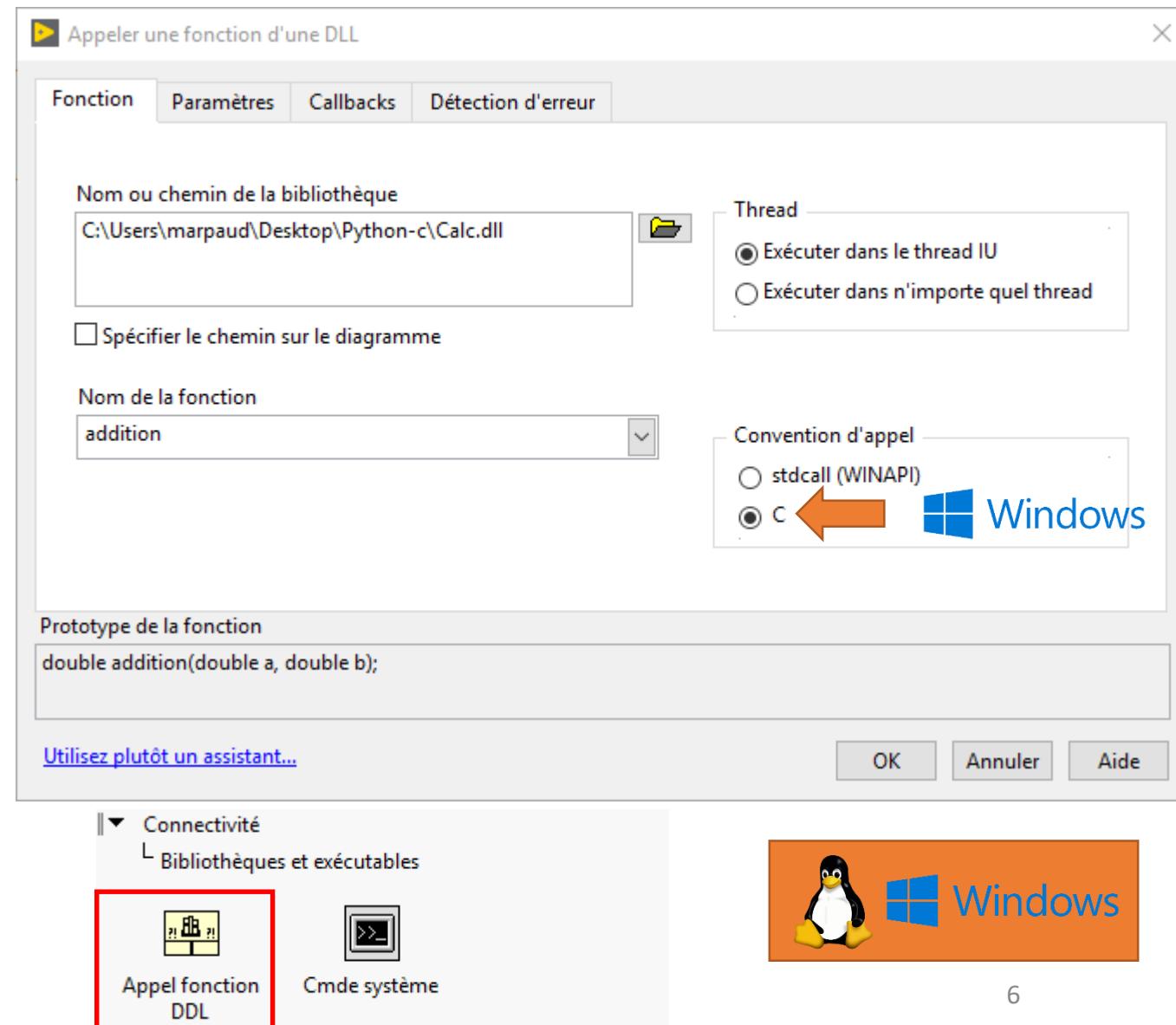
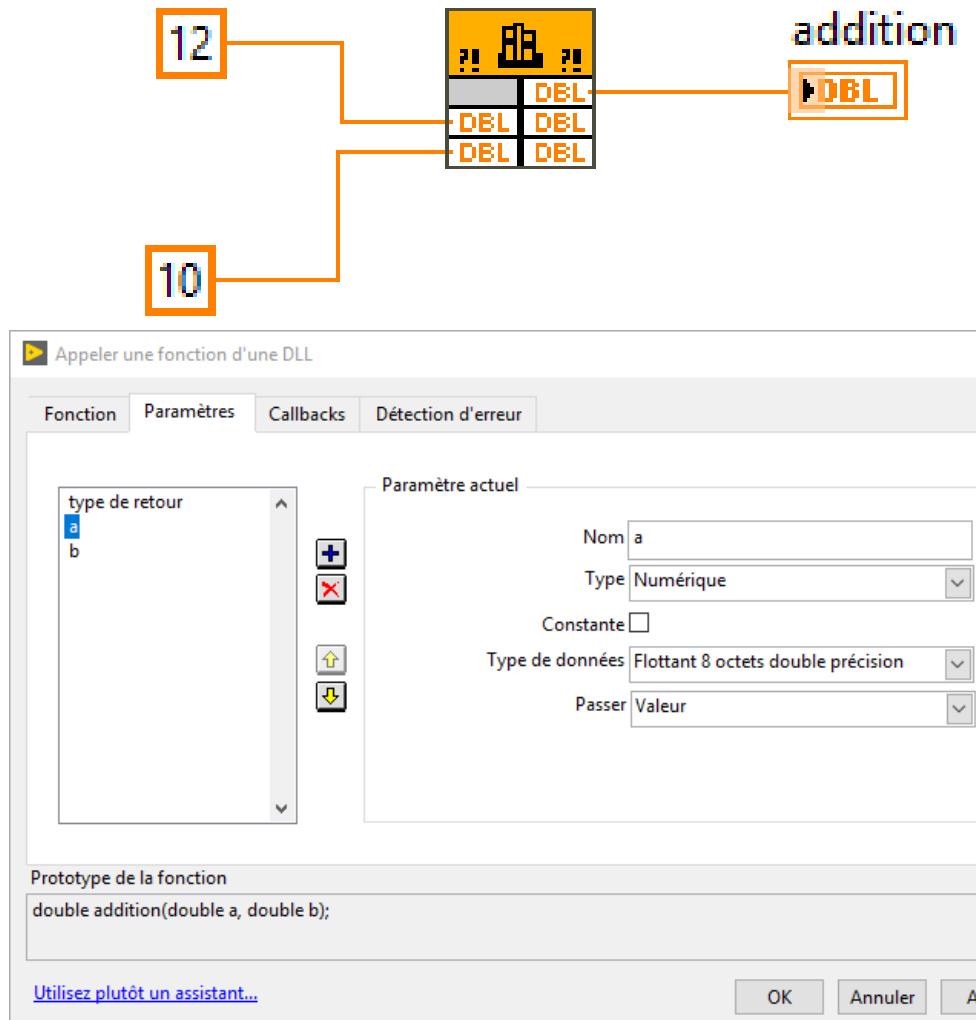
```
mjulien@localhost:~/Documents/Python-so/cc
[mjulien@localhost cc]$ gcc -c -fpic Calc.c
[mjulien@localhost cc]$ gcc -shared *.o -o Calc.so
[mjulien@localhost cc]$
```

The code editor window shows the C source code for the 'Calc.c' file, which contains two functions: 'addition' and 'multiplication'.

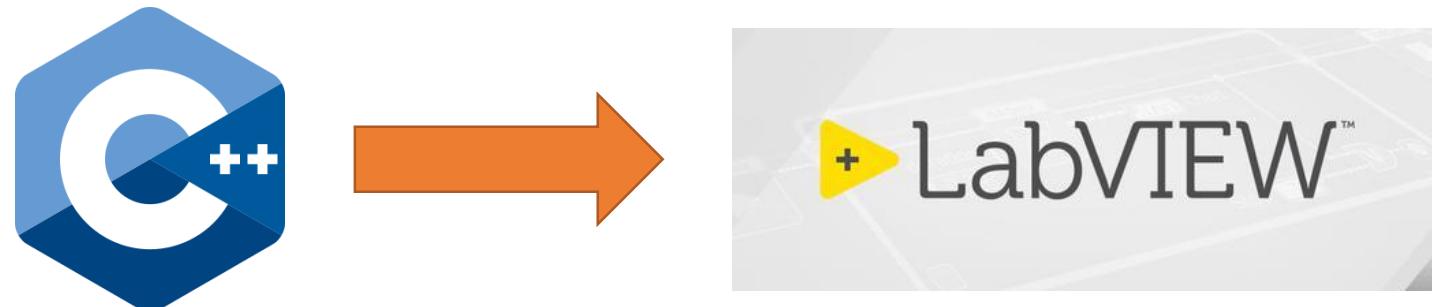
```
Calc.c
double addition(double a, double b)
{
    return a+b;
}

double multiplication(double a, double b)
{
    return a*b;
}
```

Fonctions C (2/2)



Des fonctions **C++** dans LabVIEW



Fonctions C++ (1/3)

The image shows two code editors side-by-side. The left editor, titled 'Calc.cpp', contains the following C++ code:

```
#include "Calc.h"

double addition(double a, double b)
{
    return a+b;
}

double multiplication(double a, double b)
{
    return a*b;
}
```

The right editor, titled 'Calc.h', contains the following C++ code:

```
extern "C"
{
    double addition(double a, double b);
    double multiplication(double a, double b);
}
```

An orange arrow points to the 'extern "C"' keyword in the 'Calc.h' file. Below the editors is an orange box containing the text 'Ajout de l'instruction « extern C »'.

Calc.cpp

```
#include "Calc.h"

double addition(double a, double b)
{
    return a+b;
}

double multiplication(double a, double b)
{
    return a*b;
}
```

Calc.h

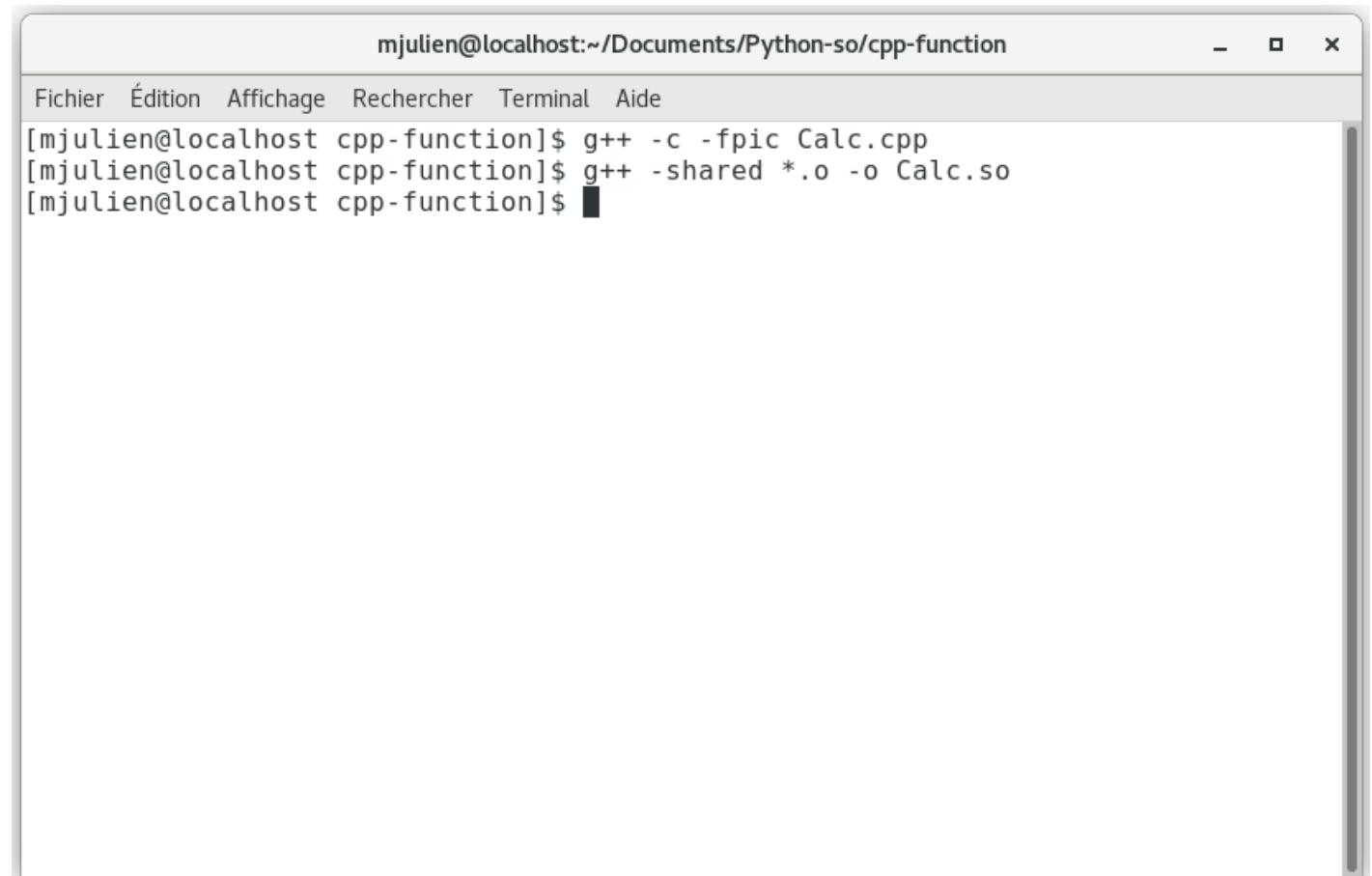
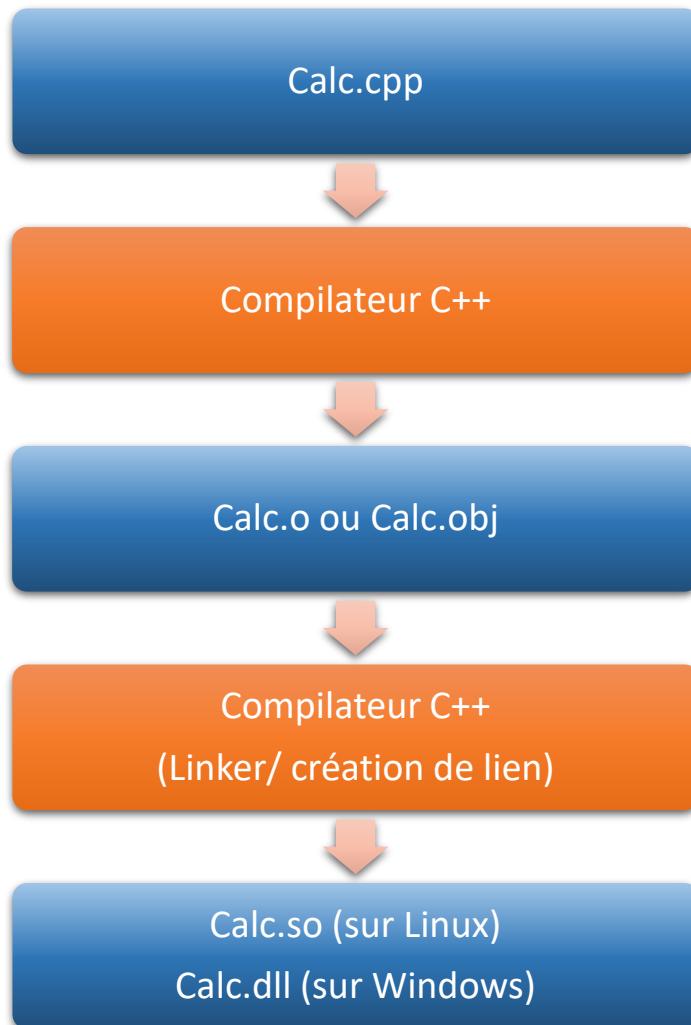
```
extern "C"
{
    double addition(double a, double b);
    double multiplication(double a, double b);
}
```

Ajout de l'instruction « extern C »

C++ ▾ Largeur des tabulations : 8 ▾ Lig 1, Col 18 ▾ INS

En-tête C/ObjC ▾ Largeur des tabulations : 8 ▾ Lig 5, Col 2 ▾ INS

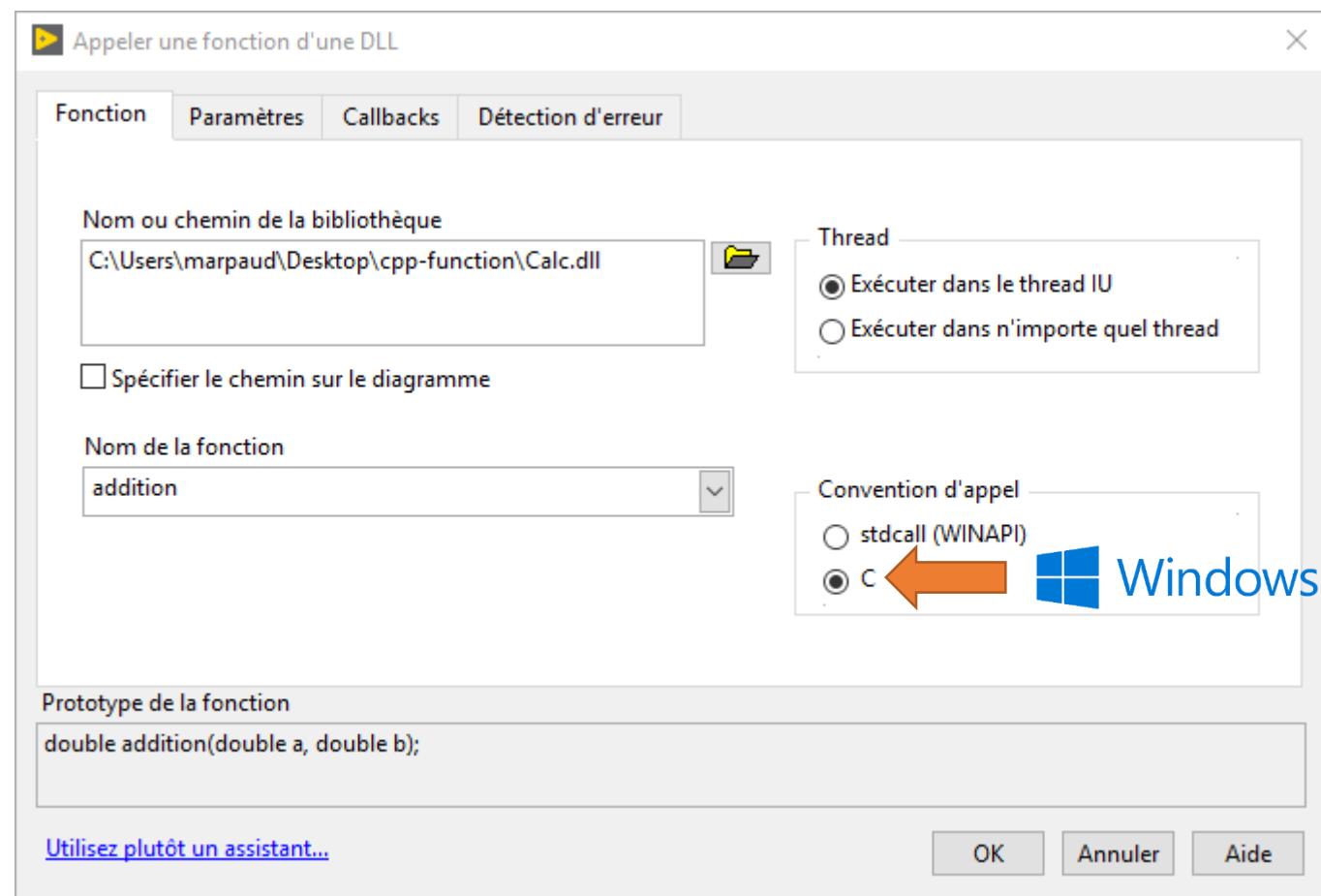
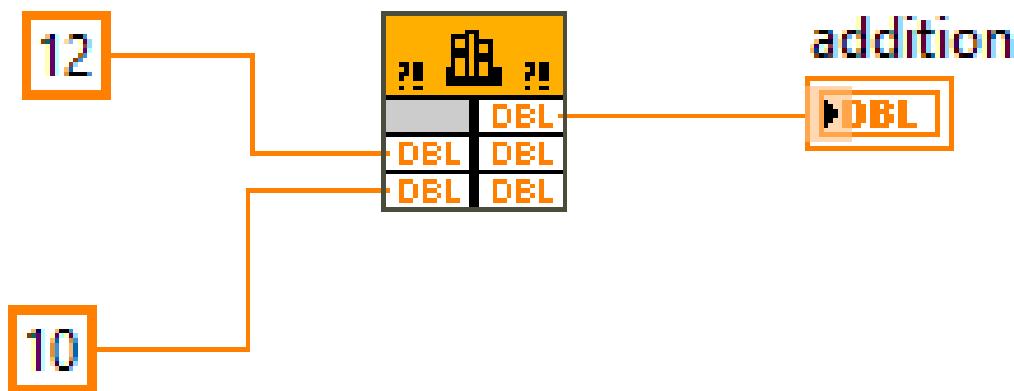
Fonctions C++ (2/3)



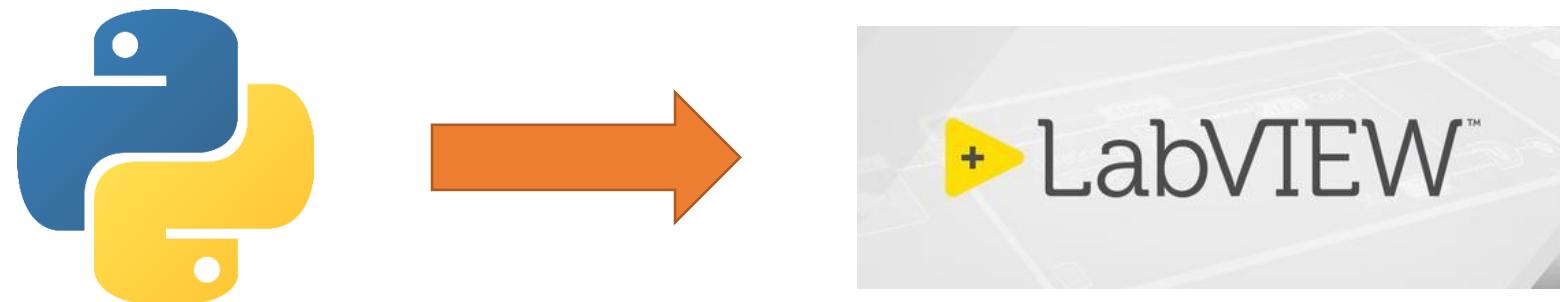
```
mjulien@localhost:~/Documents/Python-so/cpp-function
Fichier Édition Affichage Rechercher Terminal Aide
[mjulien@localhost cpp-function]$ g++ -c -fpic Calc.cpp
[mjulien@localhost cpp-function]$ g++ -shared *.o -o Calc.so
[mjulien@localhost cpp-function]$
```

A screenshot of a terminal window titled 'mjulien@localhost:~/Documents/Python-so/cpp-function'. The window shows the command line interface for compiling a C++ program. The user runs 'g++ -c -fpic Calc.cpp' to compile the source file into an object file. Then, they run 'g++ -shared *.o -o Calc.so' to link the object files into a shared library named 'Calc.so'. The terminal window includes a menu bar with 'Fichier', 'Édition', 'Affichage', 'Rechercher', 'Terminal', and 'Aide'.

Fonctions C++ (3/3)

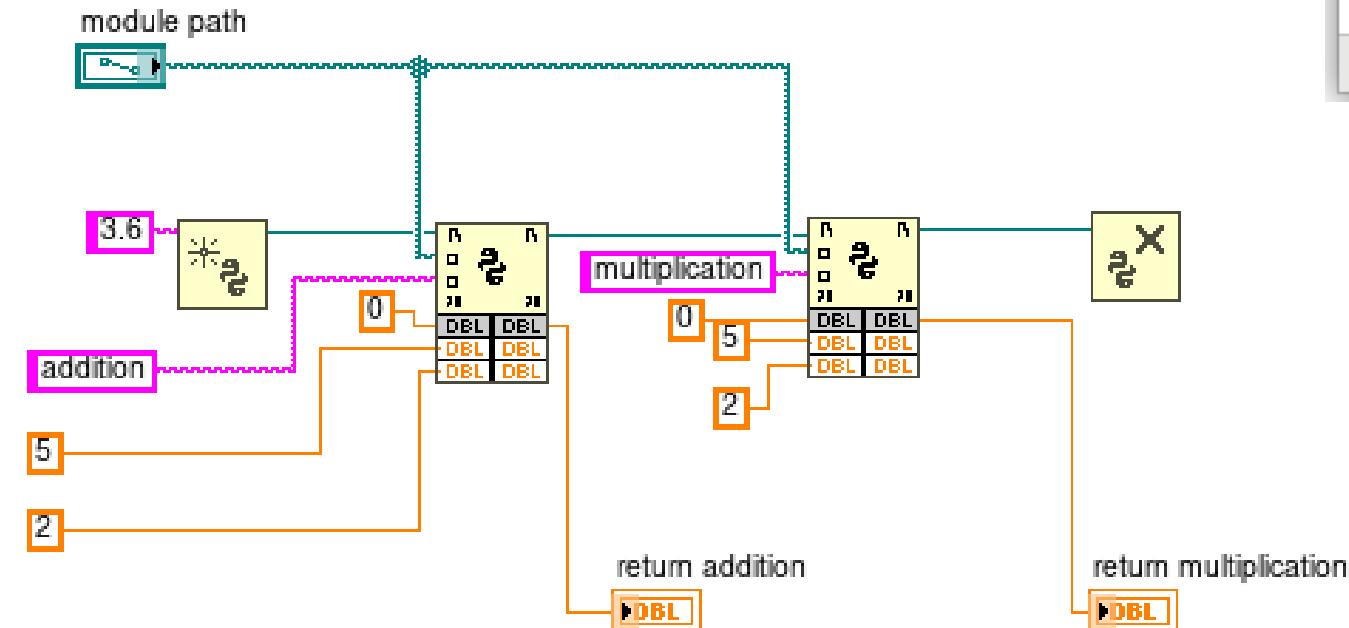


Fonctions **Python** dans LabVIEW



Rq: à partir de LabVIEW 2018

Fonctions Python (1/1)

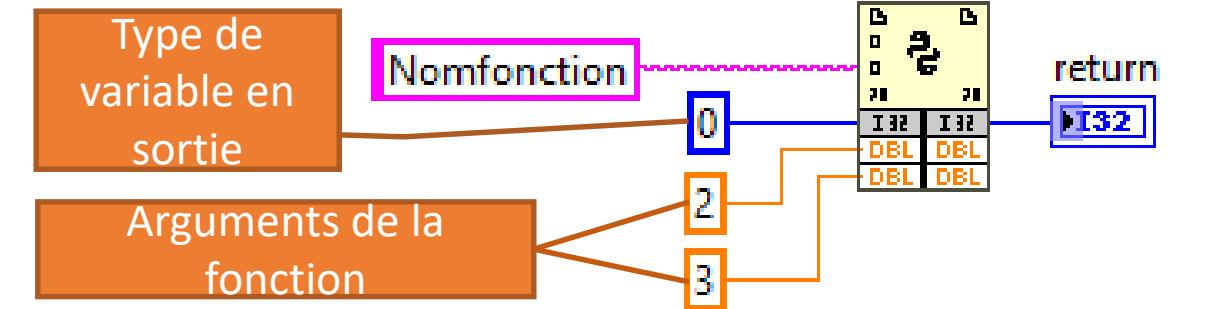


Screenshot of a Python code editor showing the following code:

```
def addition(a, b):
    return a+b

def multiplication(a, b):
    return a*b
```

Python ▾ Largeur des tabulations : 8 ▾ Lig 5, Col 19 ▾ INS



Connectivité
Python



Ouvrir session
Python



Nœud Python



Fermer session
Python

Go dans LabVIEW

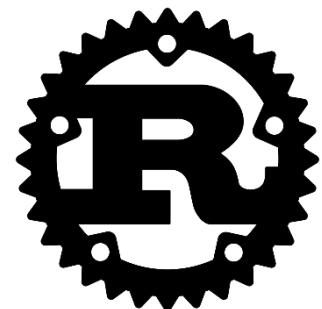


Go dans LabVIEW (1/1)

The image shows a LabVIEW session with three main windows:

- Terminal Window:** Shows a Linux terminal session with the command `go build -buildmode=c-shared -o test_go.so test.go` highlighted in red.
- Code Editor:** Shows a Go code file named `test.go` with annotations:
 - `import "C"` is highlighted in red.
 - Comments `//export addition` and `//export multiplication` are preceded by orange arrows pointing to the right.
- Block Diagram:** Shows a LabVIEW block diagram titled "test-go.vi Block Diagram" with two subroutines:
 - `addition`: Takes two inputs (12, 5) and produces one output (DBL).
 - `multiplication`: Takes two inputs (12, 5) and produces one output (DBL).

Rust dans LabVIEW



Rust dans LabVIEW (1/1)



```
Ouvrir lib.rs ~/Documents/rust/Ca... Enregistrer - x
#[no_mangle]
pub extern fn addition(nb1: f64, nb2: f64) -> f64 {
    return nb1 + nb2;
}

#[no_mangle]
pub extern fn multiplication(nb1: f64, nb2: f64) -> f64 {
    return nb1 * nb2;
}
```

```
Ouvrir Cargo.toml ~/Documents/rust/Calc Enregistrer - x
[package]
name = "Calc"
version = "0.1.0"
authors = ["mjulien"]
edition = "2018"

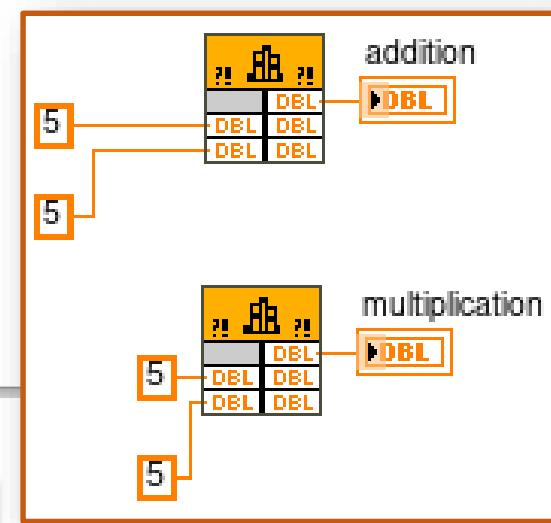
# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]

[lib]
name = "Calc"
crate-type = ["dylib"]
```

```
mjulien@localhost:~/Documents/rust/Calc
Fichier Édition Affichage Rechercher Terminal Aide
[mjulien@localhost Calc]$ cargo build --release
Compiling Calc v0.1.0 (/home/mjulien/Documents/rust/Calc)
warning: crate `Calc` should have a snake case name
  = note: `#[warn(non_snake_case)]` on by default
  = help: convert the identifier to snake case: `calc`

    Finished release [optimized] target(s) in 0.60s
[mjulien@localhost Calc]$
```



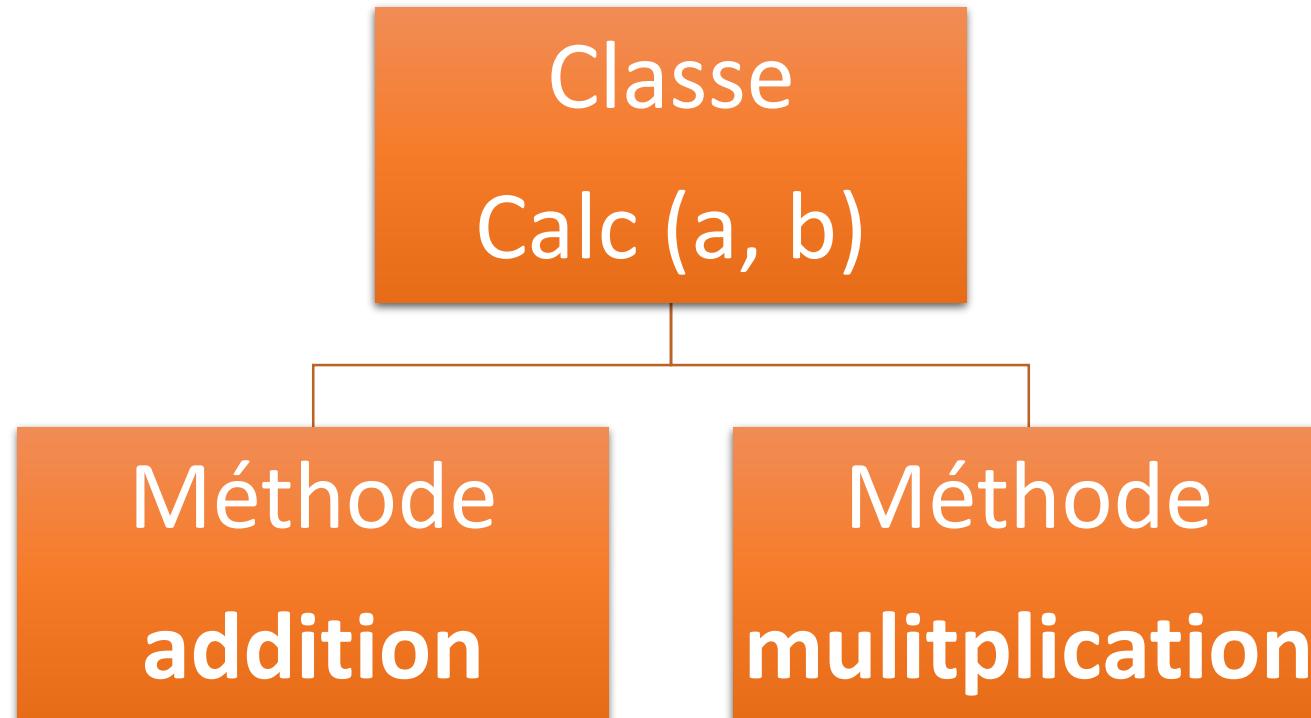
Voir les annexes pour Cargo et
Rust

II. Interfaçage d'objet

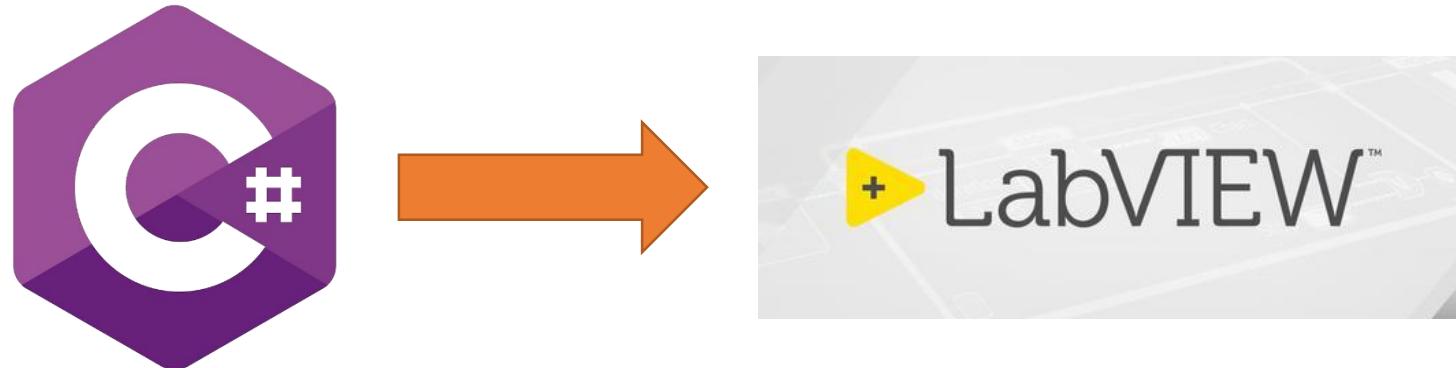
1. Objet
2. C#
3. Python
4. C++ vers Python
5. C++

Objet (1/1)

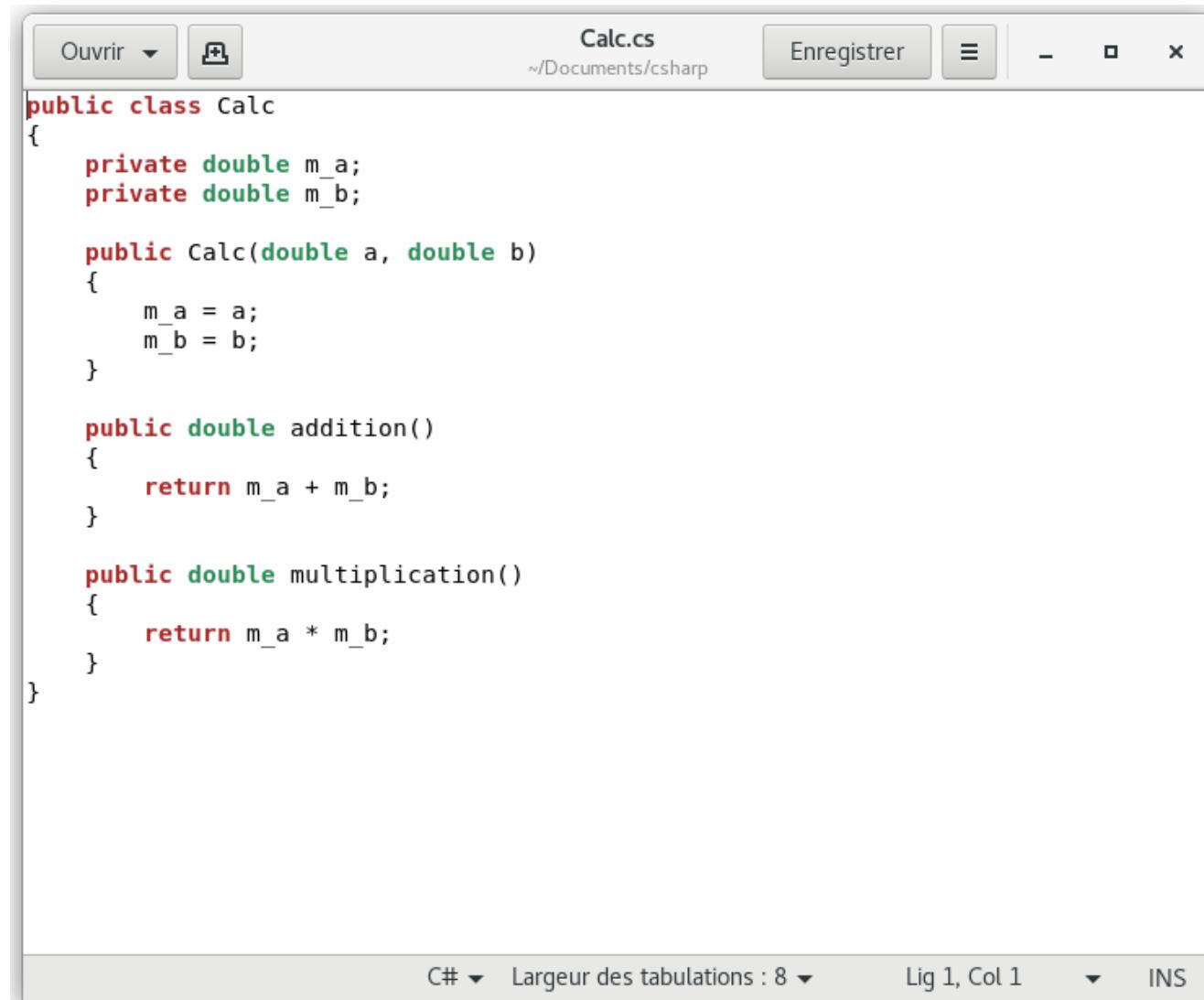
- Une classe **Calc** avec:
 - Une méthode **addition**
 - Une méthode **multiplication**



Objet **C#** dans LabVIEW



Objet C# dans LabVIEW (1/3)



The screenshot shows a C# code editor window with the following code:

```
public class Calc
{
    private double m_a;
    private double m_b;

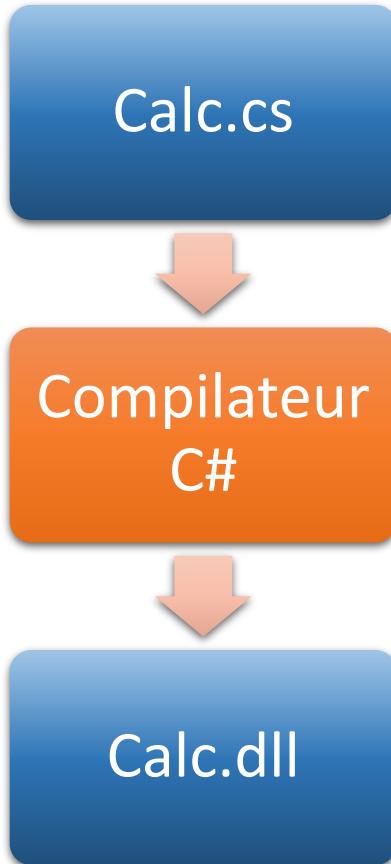
    public Calc(double a, double b)
    {
        m_a = a;
        m_b = b;
    }

    public double addition()
    {
        return m_a + m_b;
    }

    public double multiplication()
    {
        return m_a * m_b;
    }
}
```

The code defines a class named `Calc` with two private fields `m_a` and `m_b`. It has a constructor that takes two `double` parameters and initializes the fields. It also has two public methods: `addition` which returns the sum of `m_a` and `m_b`, and `multiplication` which returns the product of `m_a` and `m_b`.

Objet C# dans LabVIEW (2/3)



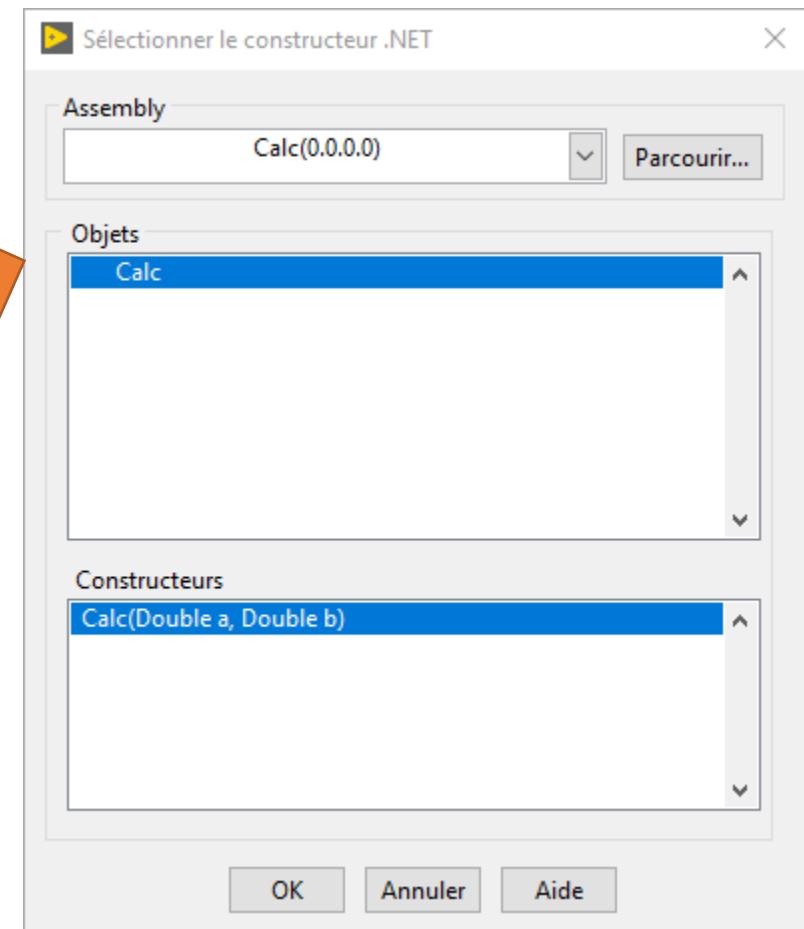
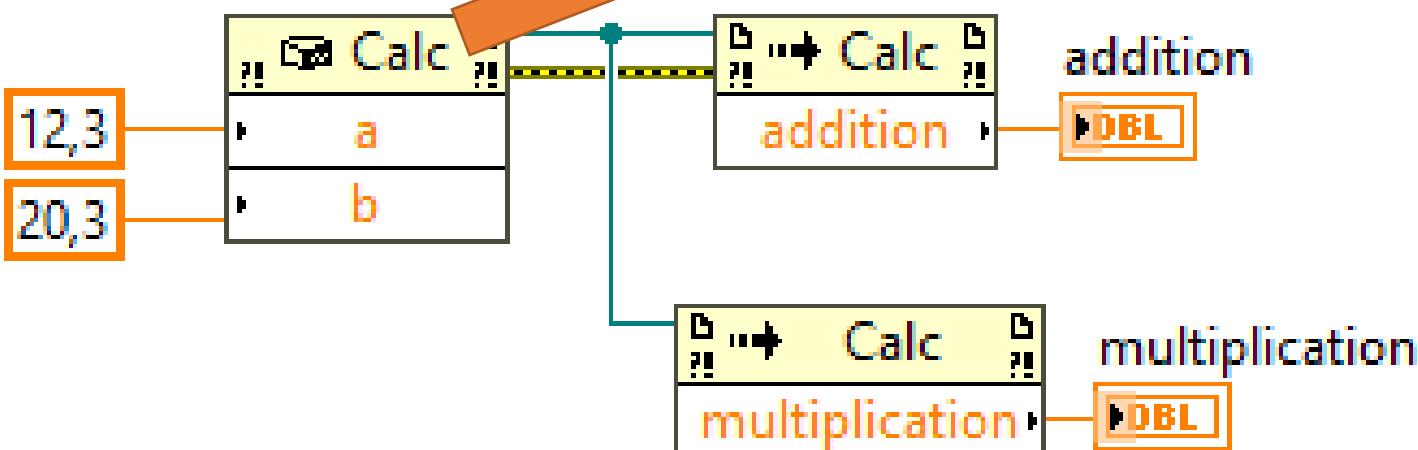
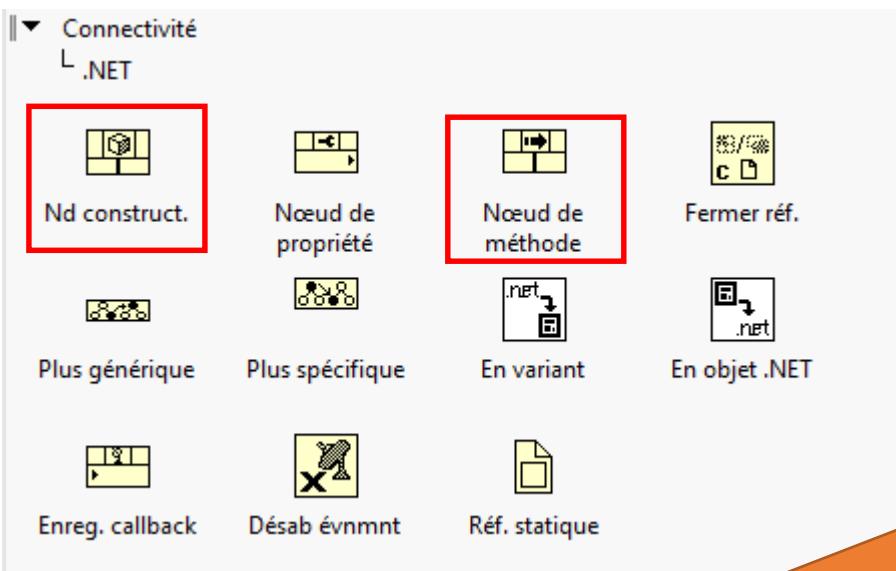
```
x86 Native Tools Command Prompt for VS 2017

C:\Users\marpaud\Desktop\cs>csc /target:library /out:Calc.dll Calc.cs
Compilateur Microsoft (R) Visual C# version 2.10.0.0 (b9fb1610)
Copyright (C) Microsoft Corporation. Tous droits réservés.

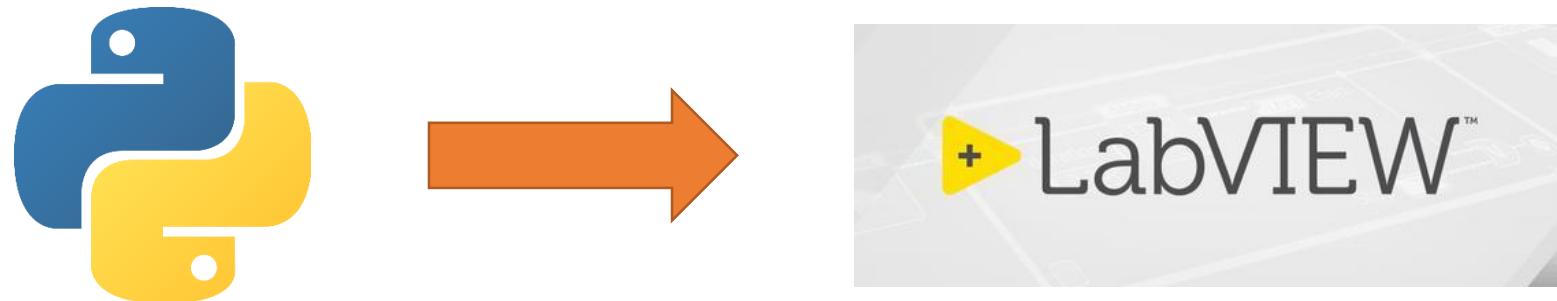
C:\Users\marpaud\Desktop\cs>
```



Objet C# dans LabVIEW (3/3)

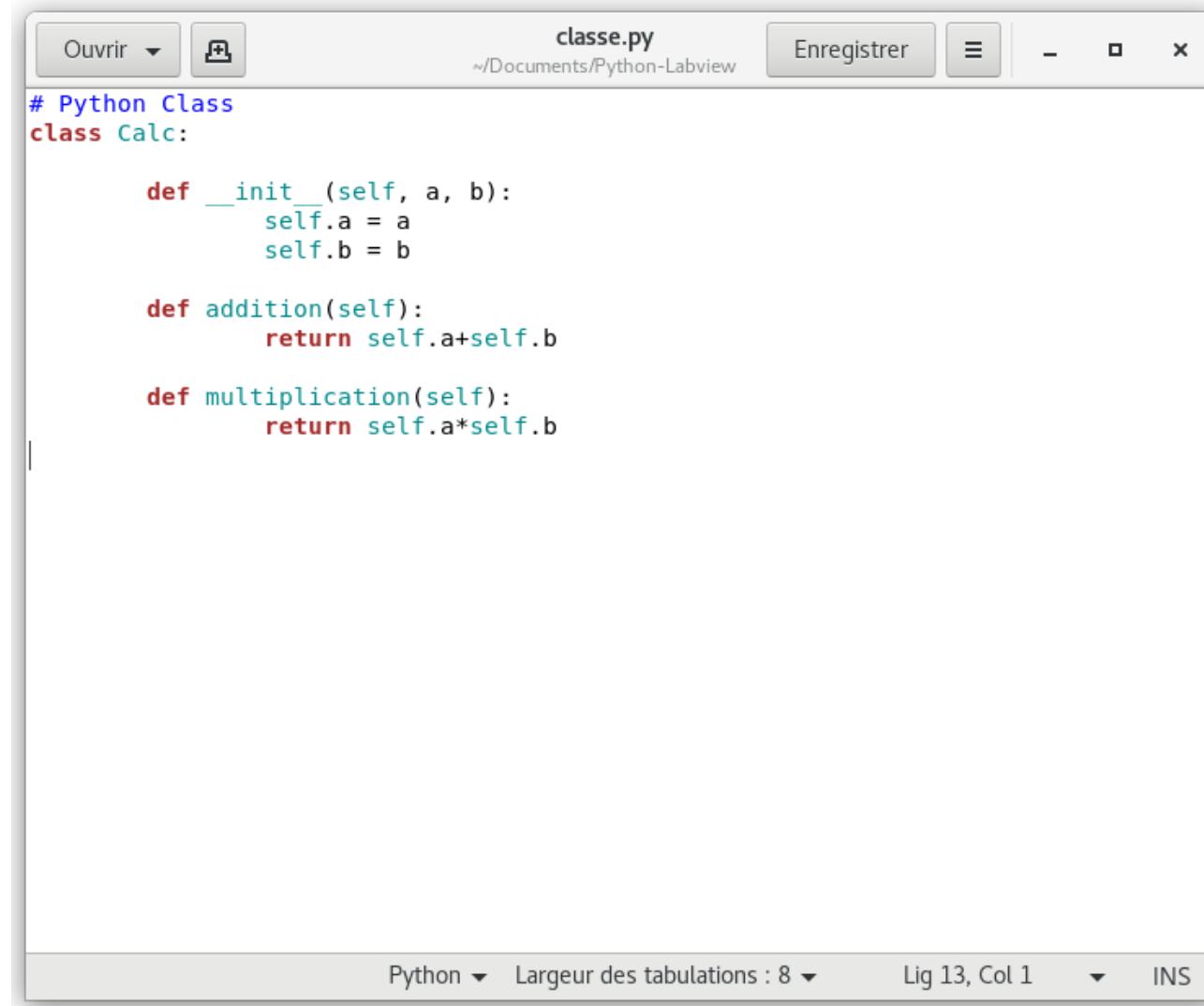


Objet Python dans LabVIEW



Rq: à partir de LabVIEW 2018

Objet Python (1/3)



The screenshot shows a Python code editor window with the following code:

```
# Python Class
class Calc:

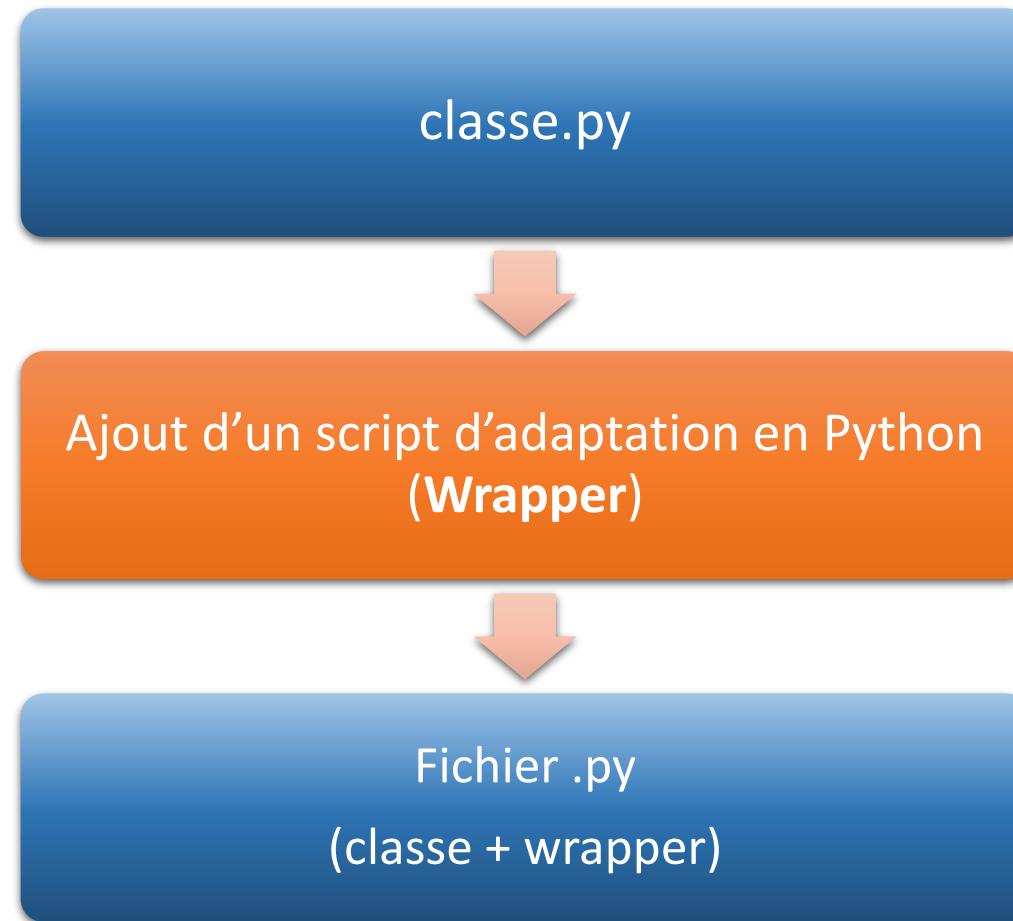
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def addition(self):
        return self.a+self.b

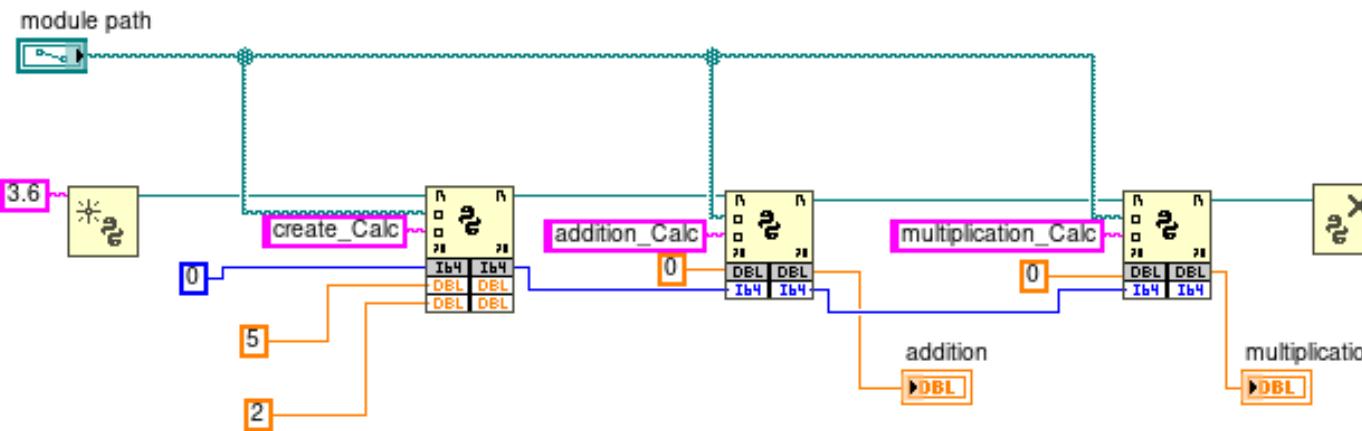
    def multiplication(self):
        return self.a*self.b
```

The window has a title bar with "classe.py" and a save button. The status bar at the bottom shows "Python" and "Largeur des tabulations : 8".

Objet **Python** (2/3)



Objet Python (3/3)



```

Ouvrir Enregistrer
classe.py
~/Documents/Python-Labview
# Python Class
class Calc:

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def addition(self):
        return self.a+self.b

    def multiplication(self):
        return self.a*self.b

# Python pour Labview
tabObject = {}
i = 0

def create_Calc(a, b):
    obj = Calc(a,b)
    global i
    global tabObject
    tabObject[i] = obj
    i = i + 1
    return i-1

def addition_Calc(index):
    global tabObject
    return tabObject[index].addition()

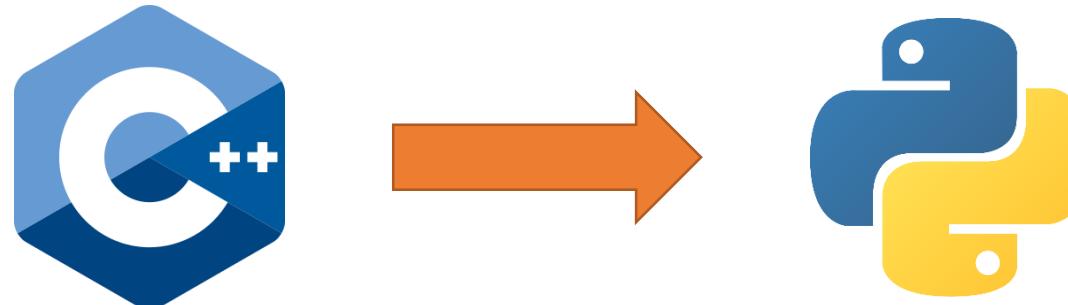
def multiplication_Calc(index):
    global tabObject
    return tabObject[index].multiplication()

def delete_Calc(index):
    global tabObject
    if index in tabObject:
        del tabObject[index]
        return True
    else:
        return False

```

C++ dans Python

(avec SWIG ou PyBind11)



C++ dans Python

- Utilisation de **SWIG** qui permet d'interfacer du C++ ou du C avec:
 - Python
 - Perl
 - C#
 - Java
 - Ruby
 - ...
- Utilisation de **PyBind11** pour interfaçer du C++ vers Python



C++ dans Python



Ouvrir Calc.cpp
~/Documents/Python-so/cpp Enregistrer -

```
#include "Calc.h"

Calc::Calc(double c, double d)
{
    a = c;
    b = d;
}

double Calc::addition()
{
    return a+b;
}

double Calc::multiplication()
{
    return a*b;
}
```

C++ Largeur des tabulations : 8 Lig 3, Col 31 INS

Ouvrir Calc.h
~/Documents/Python-so/cpp Enregistrer -

```
class Calc
{
public:
    Calc(double c, double d);
    double addition();
    double multiplication();

private:
    double a;
    double b;
};
```

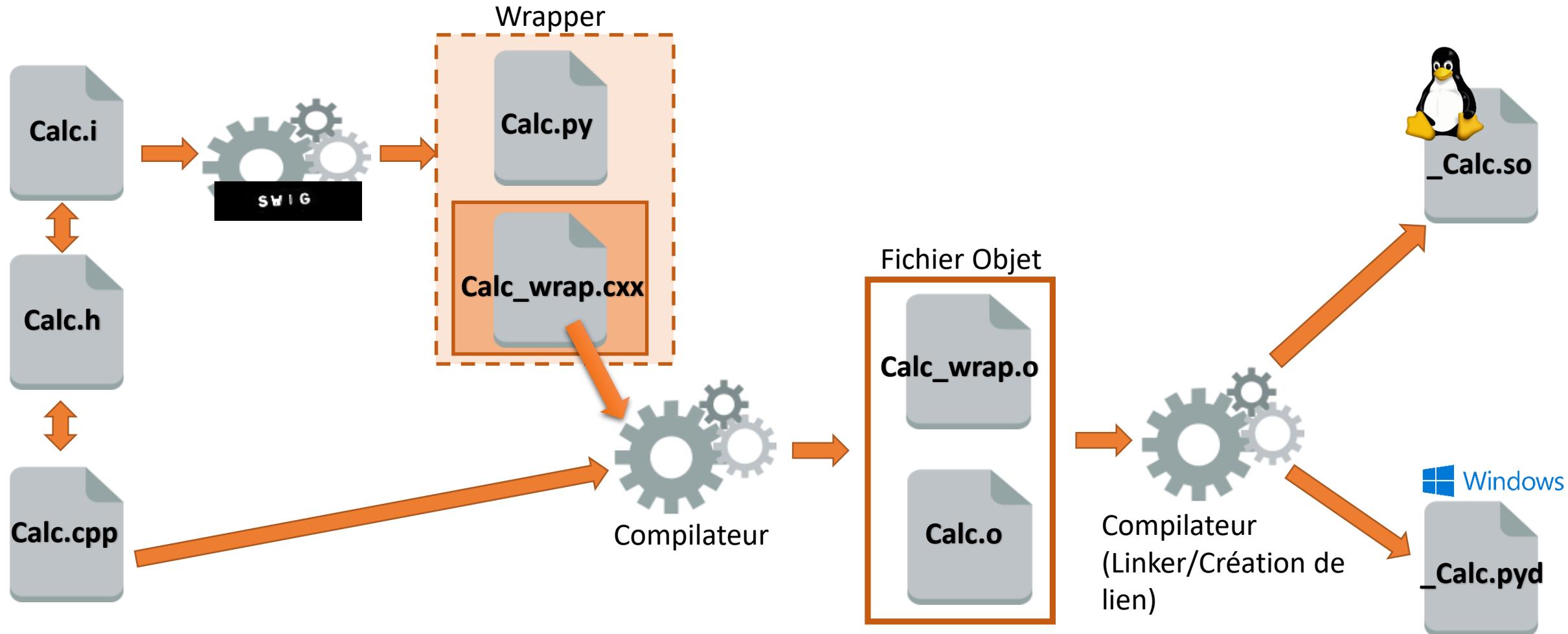
En-tête C/ObjC Largeur des tabulations : 8 Lig 1, Col 1 INS

C++ dans Python

SWIG
(Simplified Wrapper and Interface Generator)

C++ dans Python (1/4)

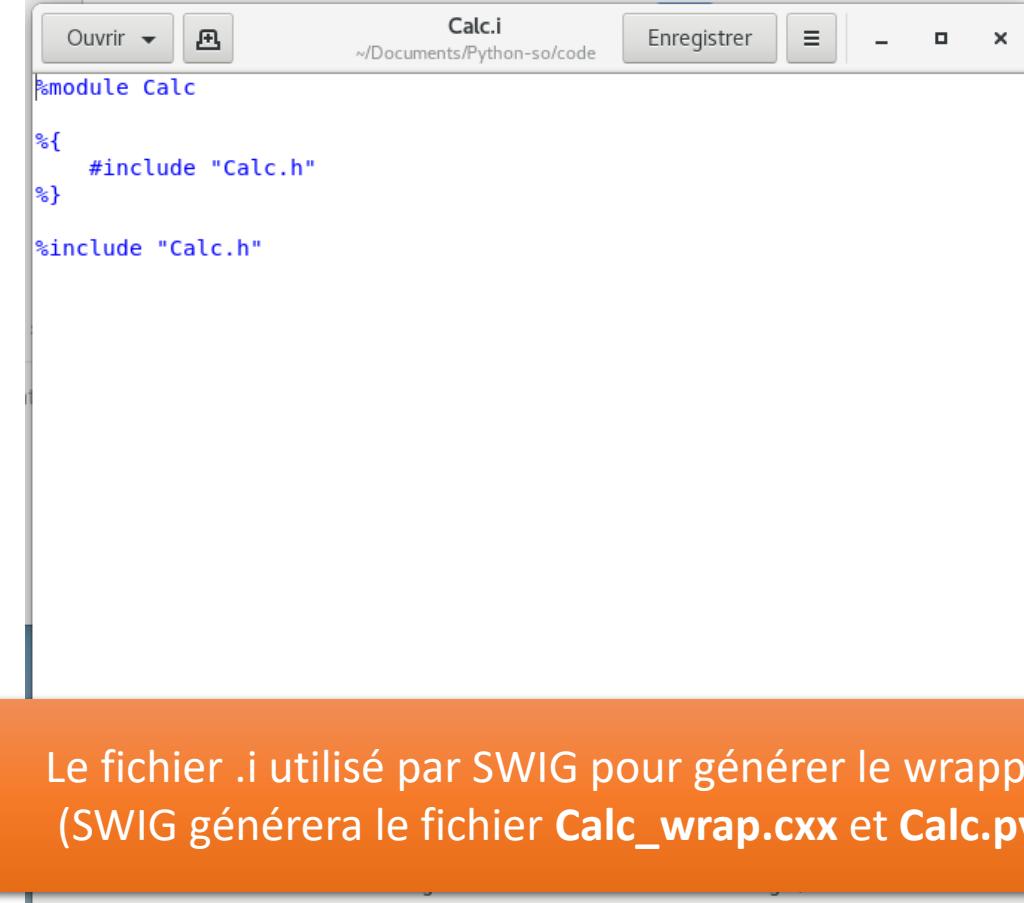
Simplified Wrapper and Interface Generator



Rq: *.pyd = *.dll compatible avec la fonction import de Python³¹

C++ dans Python (2/4)

Simplified Wrapper and Interface Generator



```
%module Calc

%{
    #include "Calc.h"
%}

#include "Calc.h"
```

Le fichier .i utilisé par SWIG pour générer le wrapper
(SWIG générera le fichier **Calc_wrap.cxx** et **Calc.py**)



C++ dans Python (3/4)

Simplified Wrapper and Interface Generator



Windows

Utilisation de MinGW
pour la compilation

```
C:\Users\marpaud\Desktop\python-swig
λ C:\Users\marpaud\Desktop\cpp-cs\swig\swig.exe -python -c++ Calc.i
```

```
C:\Users\marpaud\Desktop\python-swig
λ g++ -c -fpic Calc_wrap.cxx Calc.cpp -I C:\Python27\include
```

```
C:\Users\marpaud\Desktop\python-swig
λ g++ -shared *.o -o _Calc.pyd -LC:\Python27\libs -lpython27
```

```
IPython 5.7.0 -- An enhanced Interactive Python.
?           -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.
```

```
In [1]: import Calc

In [2]: obj = Calc.Calc(12.3,2.5)

In [3]: obj.addition()
Out[3]: 14.8
```

```
In [4]: obj.multiplication()
Out[4]: 30.75
```

C++ dans Python (4/4)

Simplified Wrapper and Interface Generator



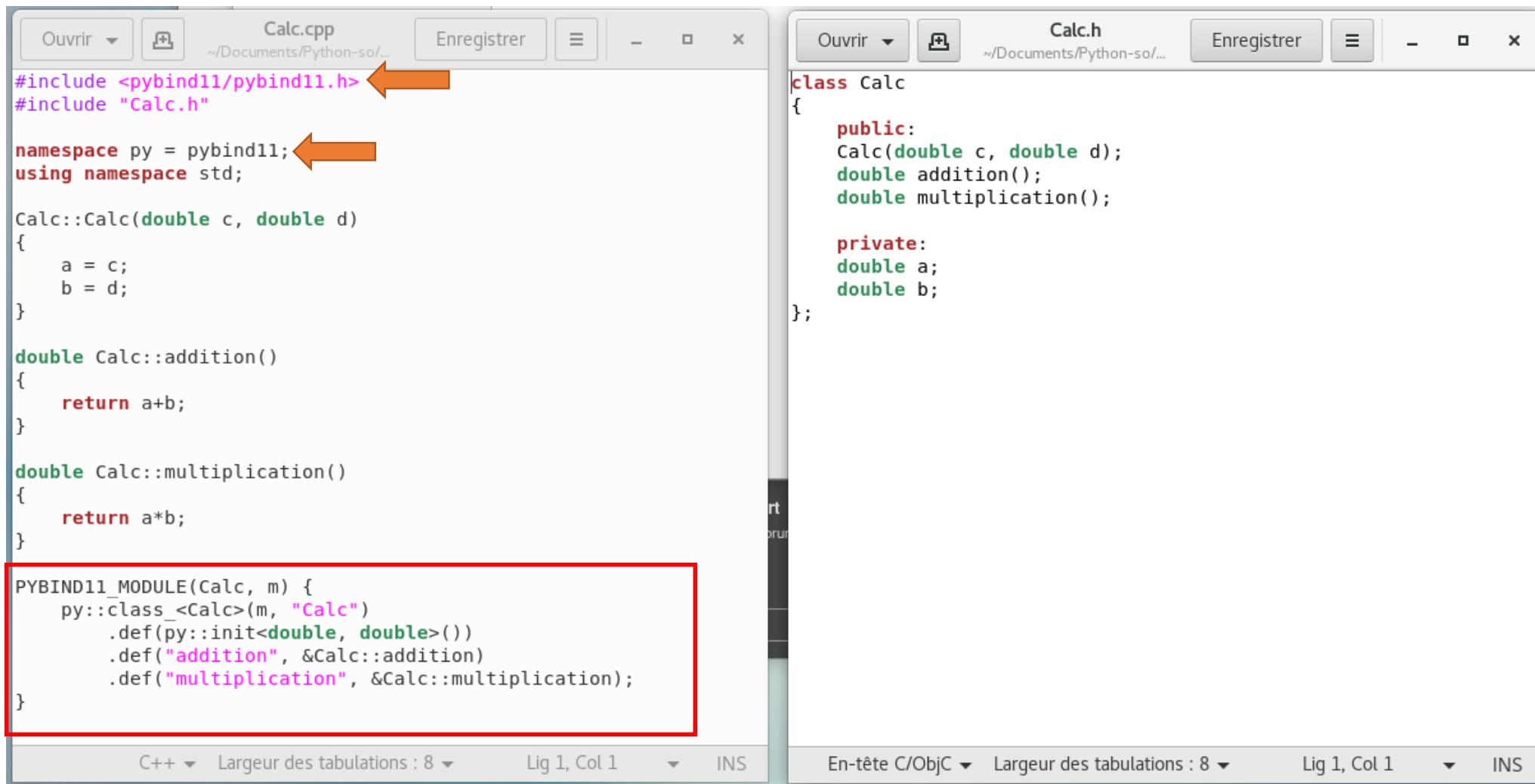
```
IPython: Python-so/code
Fichier Édition Affichage Rechercher Terminal Aide
[mjulien@localhost code]$ swig -python -c++ Calc.i
[mjulien@localhost code]$ g++ -c -fpic Calc_wrap.cxx Calc.cpp -I/usr/include/python3.6m
[mjulien@localhost code]$ g++ -shared *.o -o _Calc.so
[mjulien@localhost code]$ ipython
Python 3.6.8 (default, Aug 7 2019, 17:28:10)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.8.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import Calc
In [2]: obj = Calc.Calc(12.3,15.6)
In [3]: obj.addition()
Out[3]: 27.9
In [4]: obj.multiplication()
Out[4]: 191.88
In [5]:
```

C++ dans Python

Avec PyBind11

C++ dans Python avec PyBind11 (1/2)



```
Calc.cpp: #include <pybind11/pybind11.h> // Line 1
Calc.cpp: #include "Calc.h" // Line 2
Calc.cpp: // Line 3
Calc.cpp: namespace py = pybind11; // Line 4
Calc.cpp: using namespace std; // Line 5
Calc.cpp: // Line 6
Calc.cpp: Calc::Calc(double c, double d) // Line 7
Calc.cpp: { // Line 8
Calc.cpp:     a = c; // Line 9
Calc.cpp:     b = d; // Line 10
Calc.cpp: } // Line 11
Calc.cpp: // Line 12
Calc.cpp: double Calc::addition() // Line 13
Calc.cpp: { // Line 14
Calc.cpp:     return a+b; // Line 15
Calc.cpp: } // Line 16
Calc.cpp: // Line 17
Calc.cpp: double Calc::multiplication() // Line 18
Calc.cpp: { // Line 19
Calc.cpp:     return a*b; // Line 20
Calc.cpp: } // Line 21
Calc.cpp: // Line 22
Calc.cpp: PYBIND11_MODULE(Calc, m) { // Line 23
Calc.cpp:     py::class_<Calc>(m, "Calc") // Line 24
Calc.cpp:         .def(py::init<double, double>()) // Line 25
Calc.cpp:         .def("addition", &Calc::addition) // Line 26
Calc.cpp:         .def("multiplication", &Calc::multiplication); // Line 27
Calc.cpp: } // Line 28
Calc.cpp: // Line 29
Calc.h: // Line 30
Calc.h: class Calc // Line 31
Calc.h: { // Line 32
Calc.h:     public: // Line 33
Calc.h:         Calc(double c, double d); // Line 34
Calc.h:         double addition(); // Line 35
Calc.h:         double multiplication(); // Line 36
Calc.h:     private: // Line 37
Calc.h:         double a; // Line 38
Calc.h:         double b; // Line 39
Calc.h: } // Line 40
```

C++ dans Python avec PyBind11 (2/2)



```
Cmder
Python 2.7.15 (v2.7.15:ca079a3ea3, Apr 30 2018, 16:22:17) [MSC v.1500 32 bit (Intel)]
Type "copyright", "credits" or "license" for more information.

IPython 5.7.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

In [1]: import Calc

In [2]: obj = Calc.Calc(12.3,2.5)

In [3]: obj.multiplication()
Out[3]: 30.75

In [4]: obj.addition()
Out[4]: 14.8
```

Rq: Installer Pybind11 en utilisant:
pip install pybind11

```
g++ -O3 -Wall -shared -std=c++11 -fpic 'python3 -m pybind11 --includes' Calc.cpp -o Calc'python3-config --extension-suffix'
```

```
g++ -c -fpic Calc.cpp -I C:\Python27\include -D_hypot=hypot
g++ -shared *.o -o Calc.pyd -LC:\Python27\libs -lpython27
```



Windows

Avec g++

```
cl Calc.cpp /EHsc /I C:\Python27\include /LD C:\Python27\libs\python27.lib
```

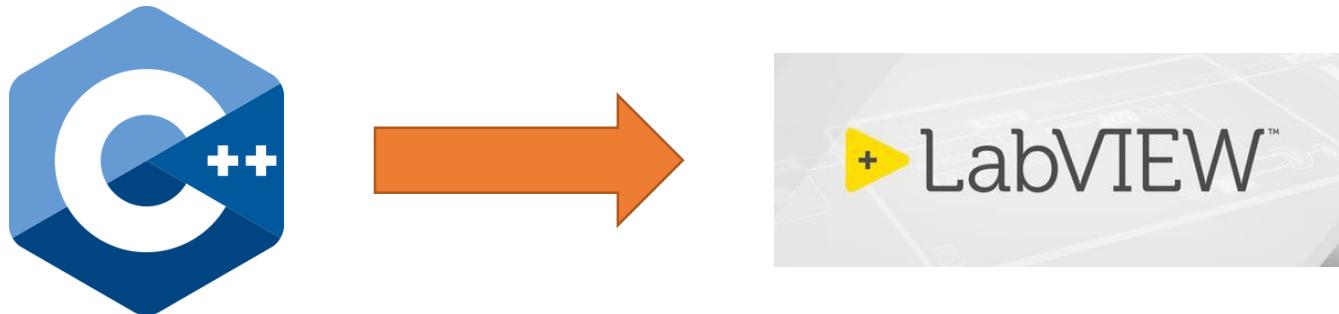
Rq: renommer Calc.dll en Calc.pyd



Windows

Avec Microsoft
Visual Studio

Objet C++ dans LabVIEW



Objet C++



```
Calc.cpp  ~/Documents/Python-so/cpp
```

```
#include "Calc.h"

Calc::Calc(double c, double d)
{
    a = c;
    b = d;
}

double Calc::addition()
{
    return a+b;
}

double Calc::multiplication()
{
    return a*b;
}
```

```
Calc.h  ~/Documents/Python-so/cpp
```

```
class Calc
{
public:
    Calc(double c, double d);
    double addition();
    double multiplication();

private:
    double a;
    double b;
};
```

C++ ▾ Largeur des tabulations : 8 ▾ Lig 3, Col 31 ▾ En-tête C/ObjC ▾ Largeur des tabulations : 8 ▾ Lig 1, Col 1 ▾ INS

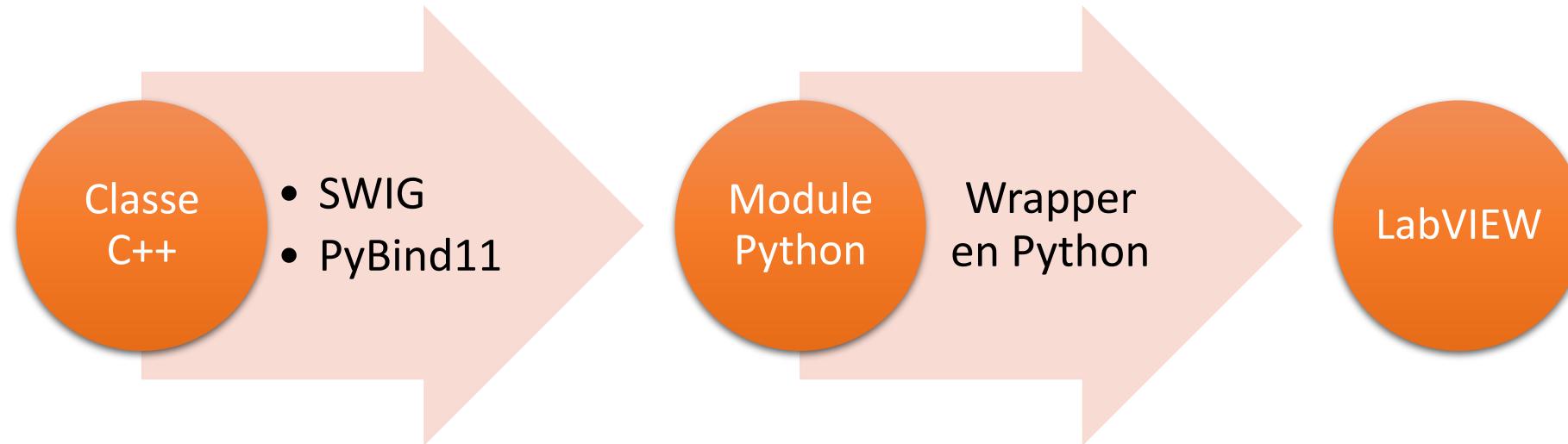
Objet C++

- Objet C++ via Python (avec PyBind11)
- Objet C++ via C# (avec SWIG)
- Crédit d'un wrapper pour interfaçer un objet C++ avec LabVIEW

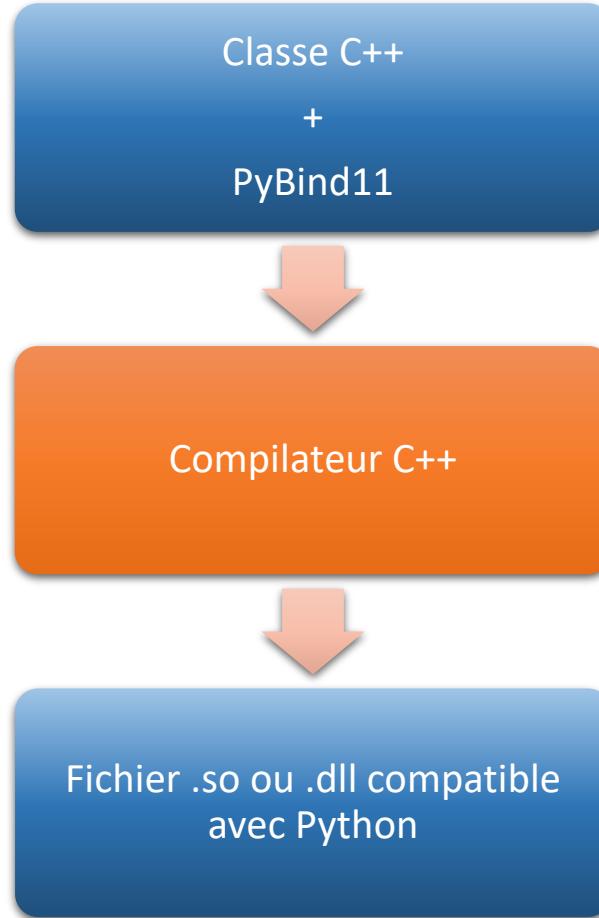
Objet C++



Objet **C++** via Python (1/3)

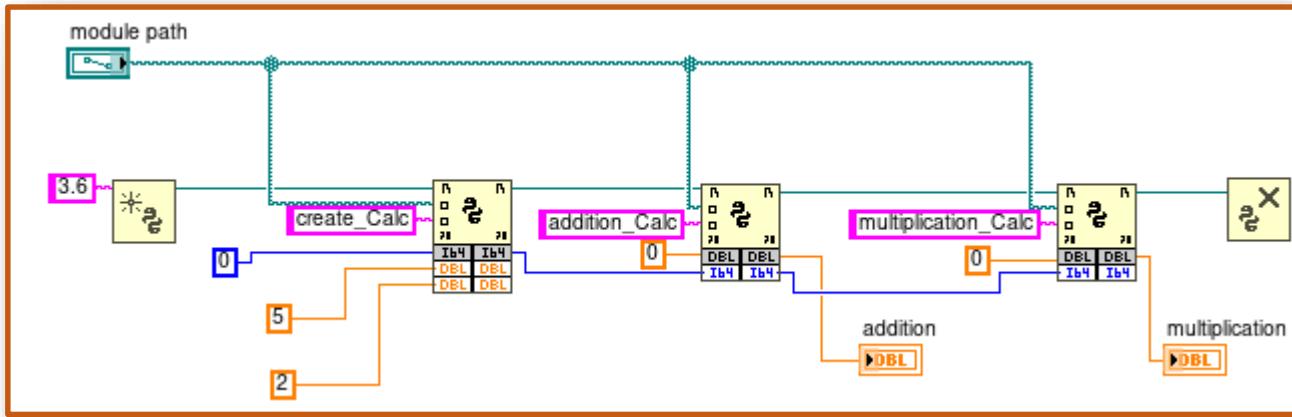


Objet C++ via Python (2/3)



Objet C++ via Python (3/3)

Appelle de la librairie
générée par PyBind11
(.dll ou .so)



```
import Calc

# Python pour Labview
tabObject = []
i = 0

def create_Calc(a, b):
    obj = Calc.Calc(a,b)
    global i
    global tabObject
    tabObject[i] = obj
    i = i + 1
    return i-1

def addition_Calc(index):
    global tabObject
    return tabObject[index].addition()

def multiplication_Calc(index):
    global tabObject
    return tabObject[index].multiplication()

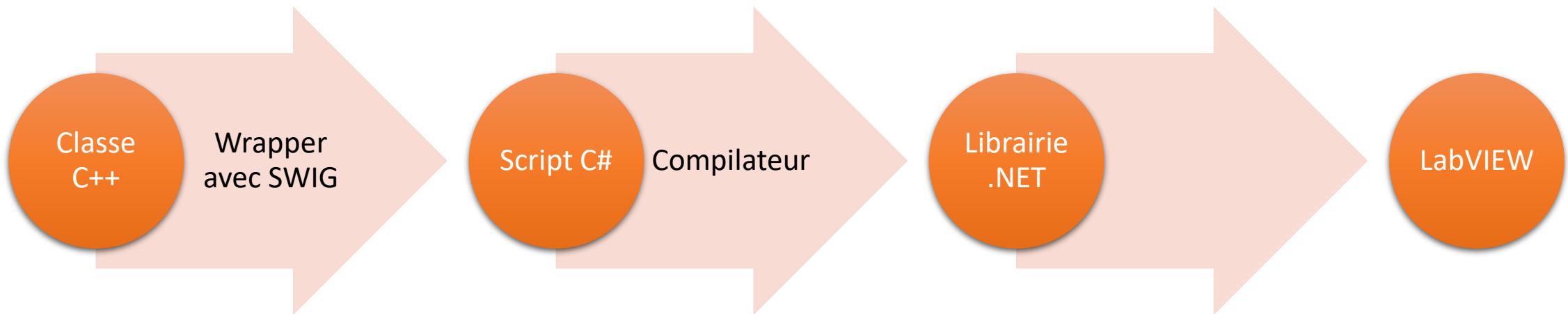
def delete_Calc(index):
    global tabObject
    if index in tabObject:
        del tabObject[index]
        return True
    else:
        return False
```



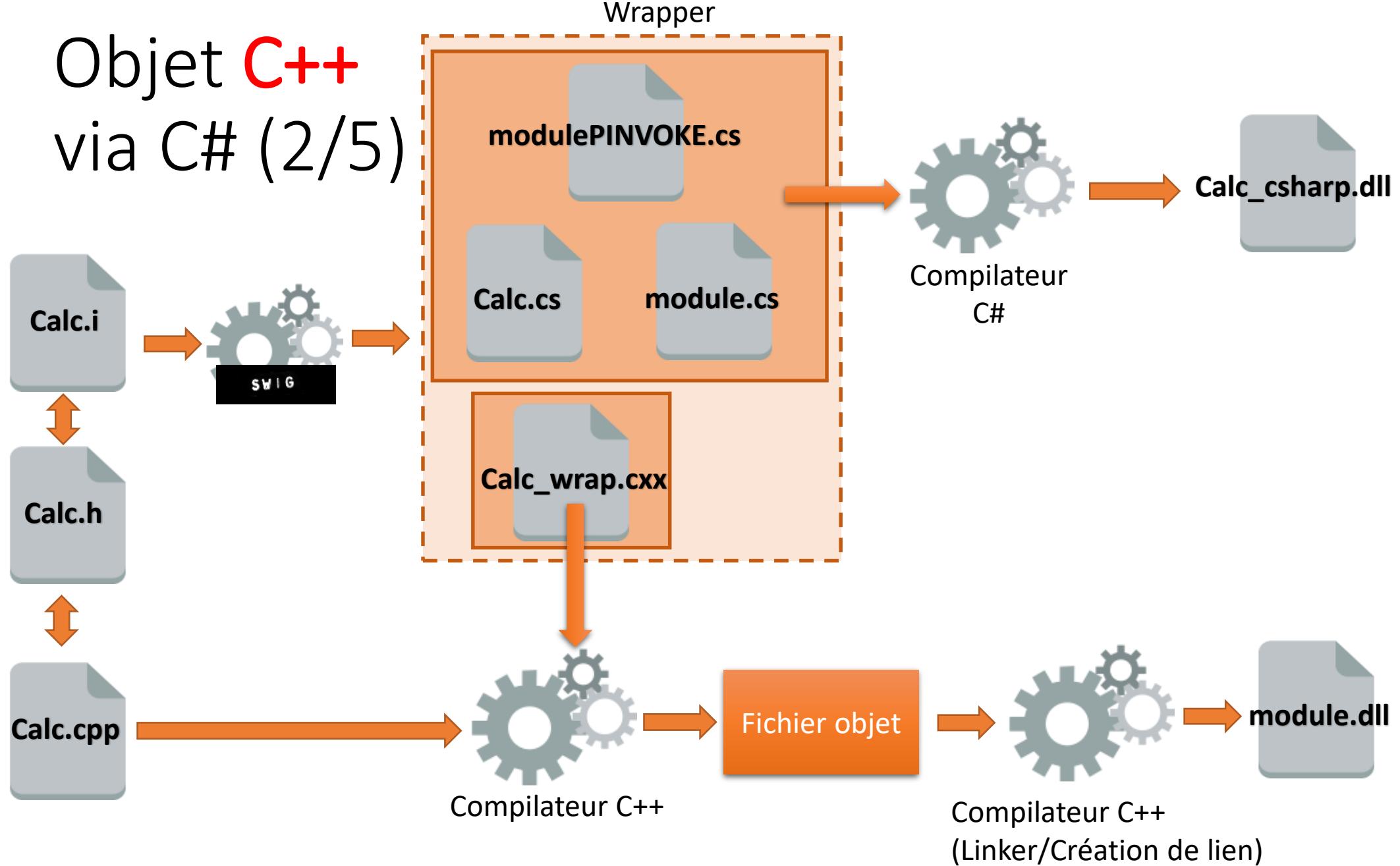
Objet C++



Objet C++ via C# (1/5)



Objet C++ via C# (2/5)



Objet C++ via C# (3/5)

Calc.i pour générer le wrapper avec SWIG



```
1 %module example
2
3 %
4     #include "Calc.h"
5 %
6
7 %include "Calc.h"
8
```

SWIG n'autorise pas que le module est le même nom que la classe en C#

Objet C++ via C# (4/5)

```
C:\Users\marpaud\Desktop\cpp-cs
λ C:\Users\marpaud\Desktop\cpp-cs\swig\swig.exe -csharp -c++ Calc.i
Visual C# Source File 1 Ko
C:\Users\marpaud\Desktop\cpp-cs
λ g++ -c -fpic Calc_wrap.cxx Calc.cpp
Fichier O 1 Ko
C:\Users\marpaud\Desktop\cpp-cs
λ g++ -shared -Wl,--add-stdcall-alias *.o -o example.dll
Extension de l'app...
C:\Users\marpaud\Desktop\cpp-cs
λ csc /target:library /out:Calc_csharp.dll *.cs
Compilateur Microsoft (R) Visual C# version 2.10.0.0 (b9fb1610)
Copyright (C) Microsoft Corporation. Tous droits réservés.

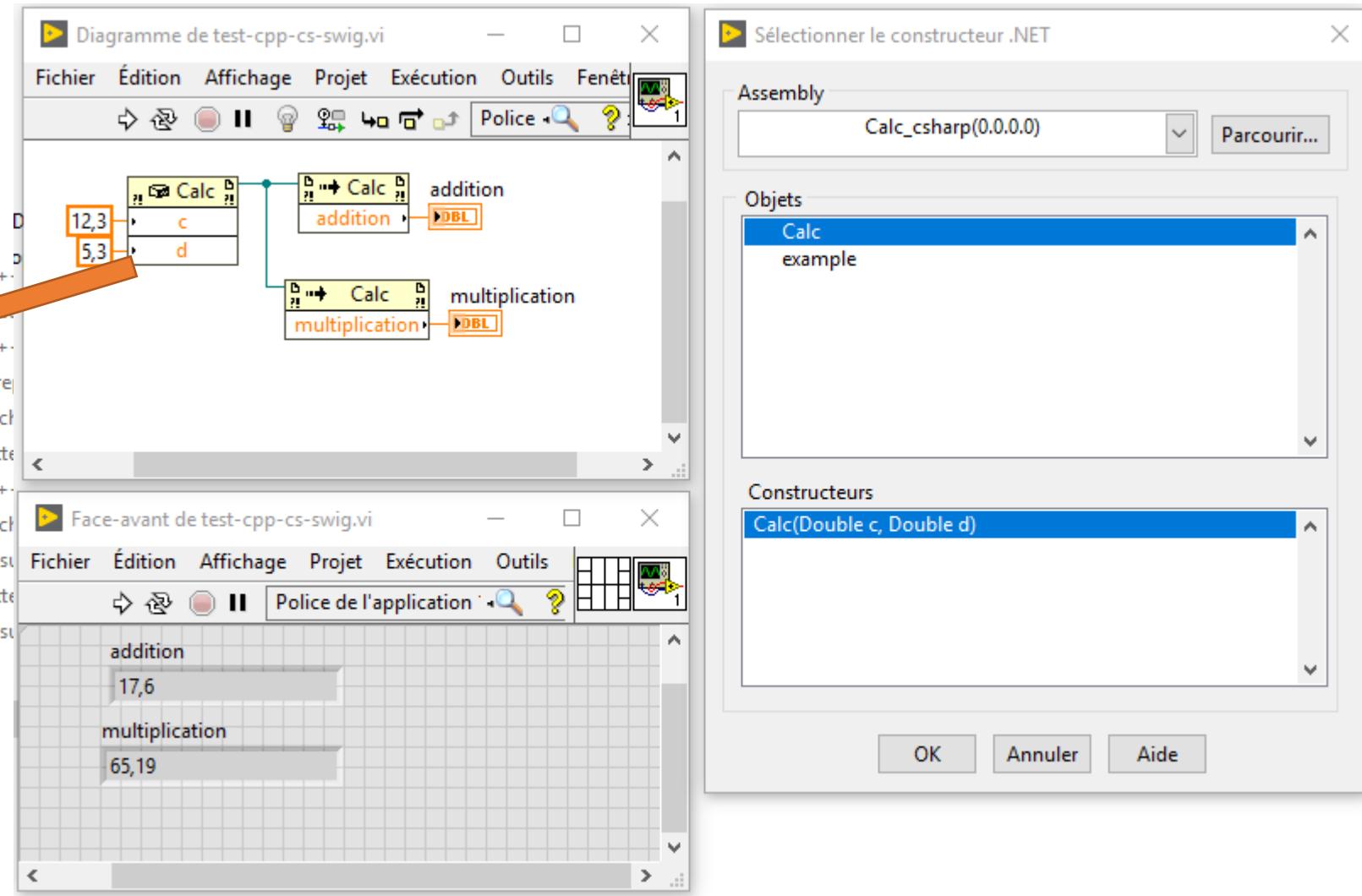
C:\Users\marpaud\Desktop\cpp-cs
```

Génération du wrapper avec
SWIG

Objet C++ via C# (5/5)



Calc.cpp
Calc.cs
Calc.h
Calc.i
Calc.o
Calc_csharp.dll
Calc_wrap.cxx
Calc_wrap.o
example.cs
example.dll
examplePINVOKE.cs

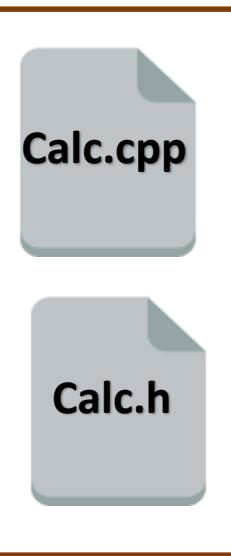


Objet C++

Création d'un wrapper pour interfaçer un objet
C++ avec LabVIEW

Objet C++

C++ avec un wrapper (1/3)



```
1 ifndef __WRAPPER_H__
2 define __WRAPPER_H__
3
4 ifdef __cplusplus
5 extern "C" {
6 endif
7
8 struct wrapper;
9 typedef struct wrapper wrapper_t;
10
11 wrapper_t *Calc_create(double a, double b);
12 void Calc_destroy(wrapper_t *m);
13
14 double Calc_addition(wrapper_t *m);
15 double Calc_multiplication(wrapper_t *m);
16
17 ifdef __cplusplus
18 }
19 endif
20
21 endif /* __WRAPPER_H__ */
```



```
#include <stdlib.h>
#include "wrapper.h"
#include "Calc.h"

struct wrapper {
    void *obj;
};

wrapper_t *Calc_create(double a, double b)
{
    wrapper_t *m;
    Calc *obj;

    m = (typeof(m)) malloc(sizeof(*m));
    obj = new Calc(a, b);
    m->obj = obj;

    return m;
}

void Calc_destroy(wrapper_t *m)
{
    if (m == NULL)
        return;
    delete static_cast<Calc *>(m->obj);
    free(m);
}

double Calc_addition(wrapper_t *m)
{
    Calc *obj;

    if (m == NULL)
        return 0;

    obj = static_cast<Calc *>(m->obj);
    return obj->addition();
}

double Calc_multiplication(wrapper_t *m)
{
    Calc *obj;

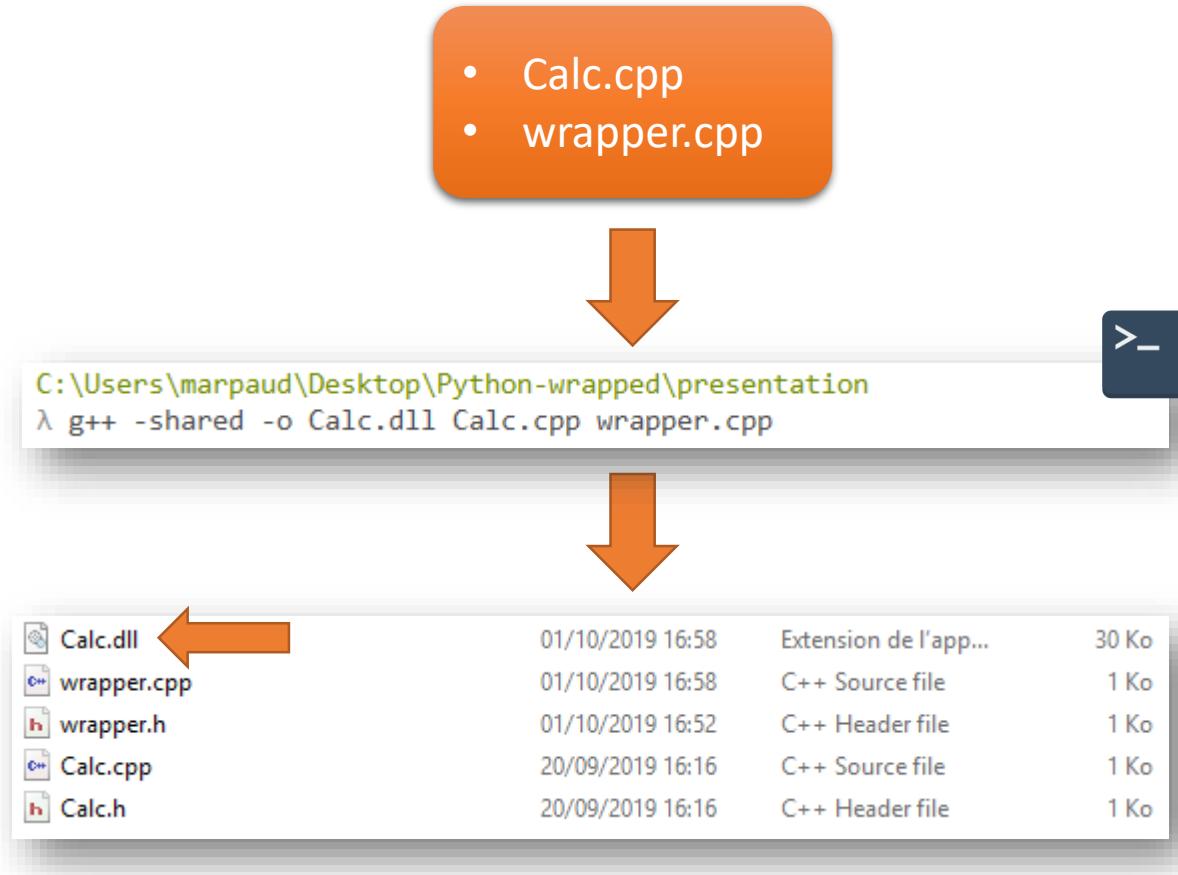
    if (m == NULL)
        return 0;

    obj = static_cast<Calc *>(m->obj);
    return obj->multiplication();
}
```



Objet C++

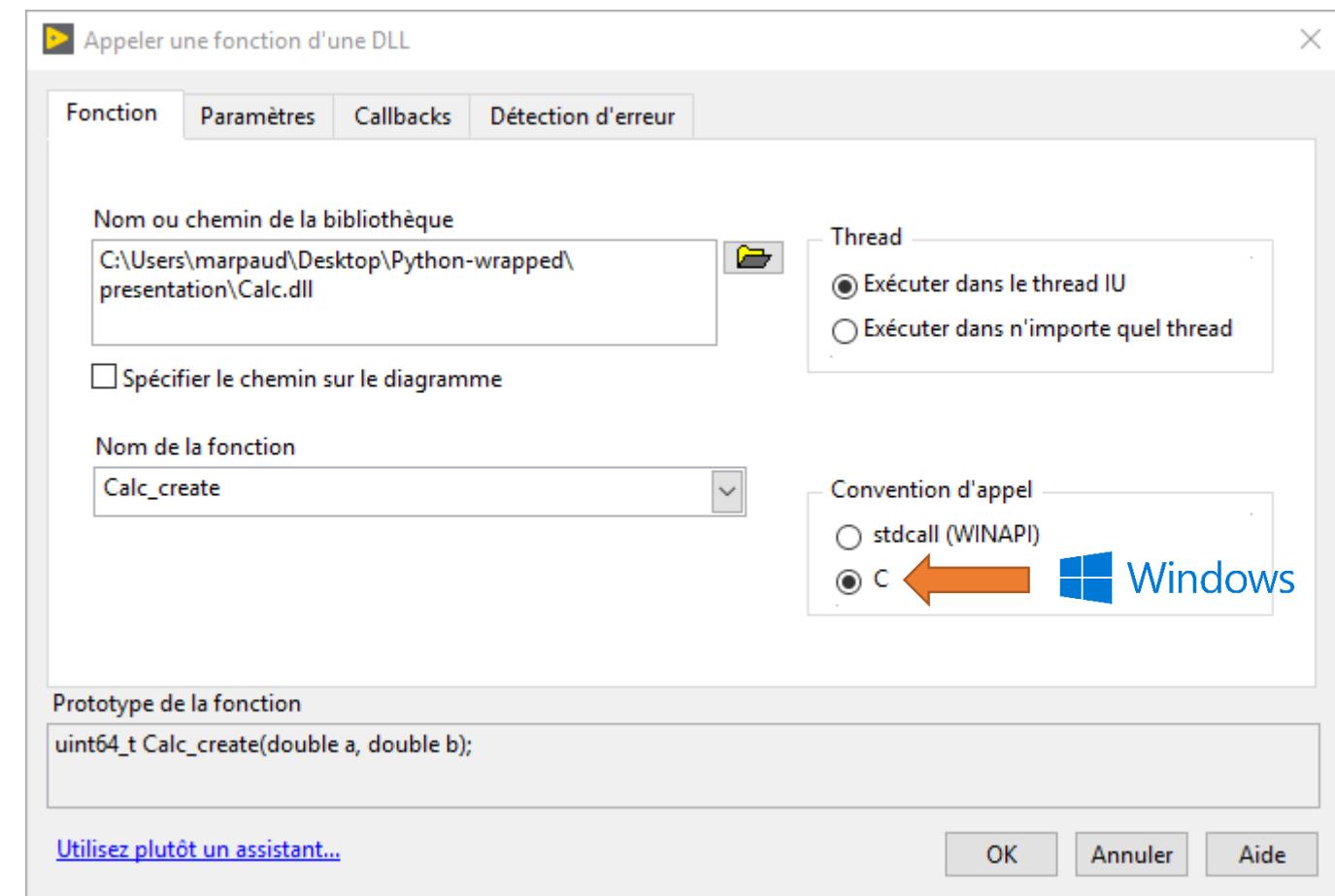
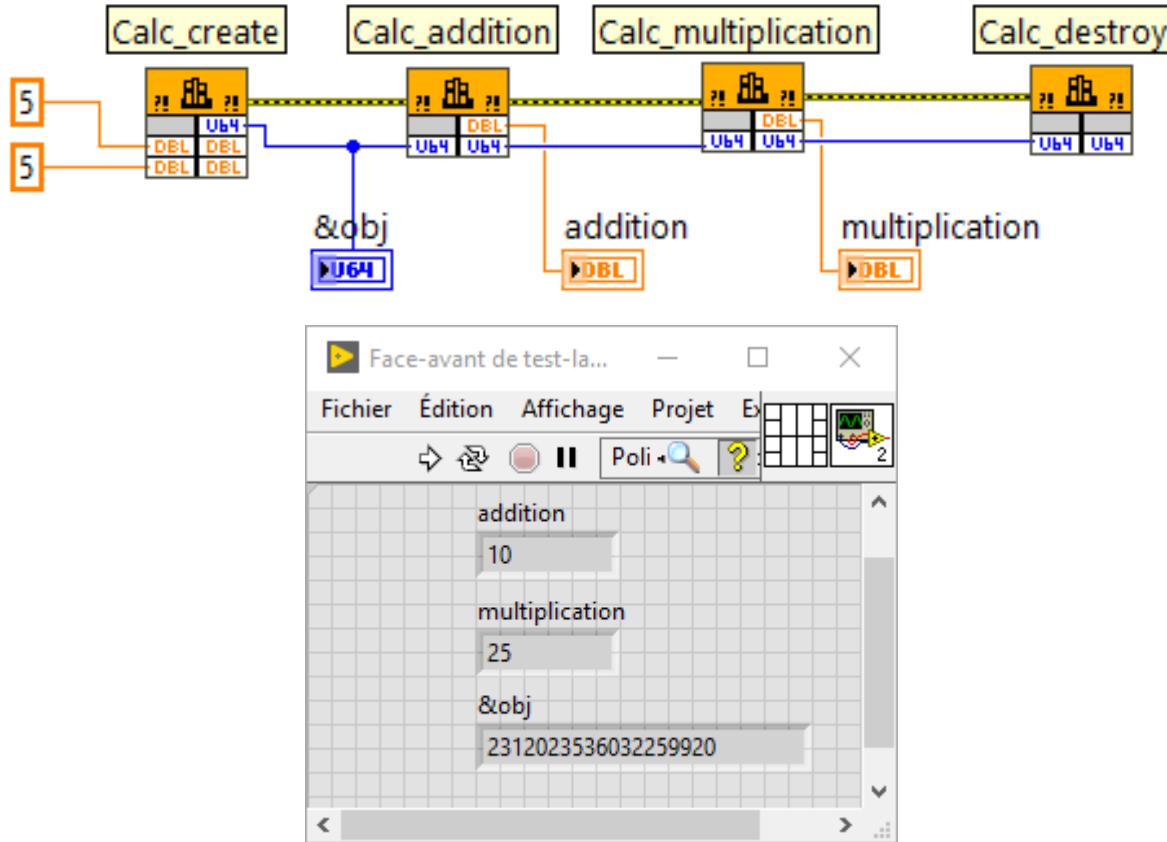
C++ avec un wrapper (2/3)



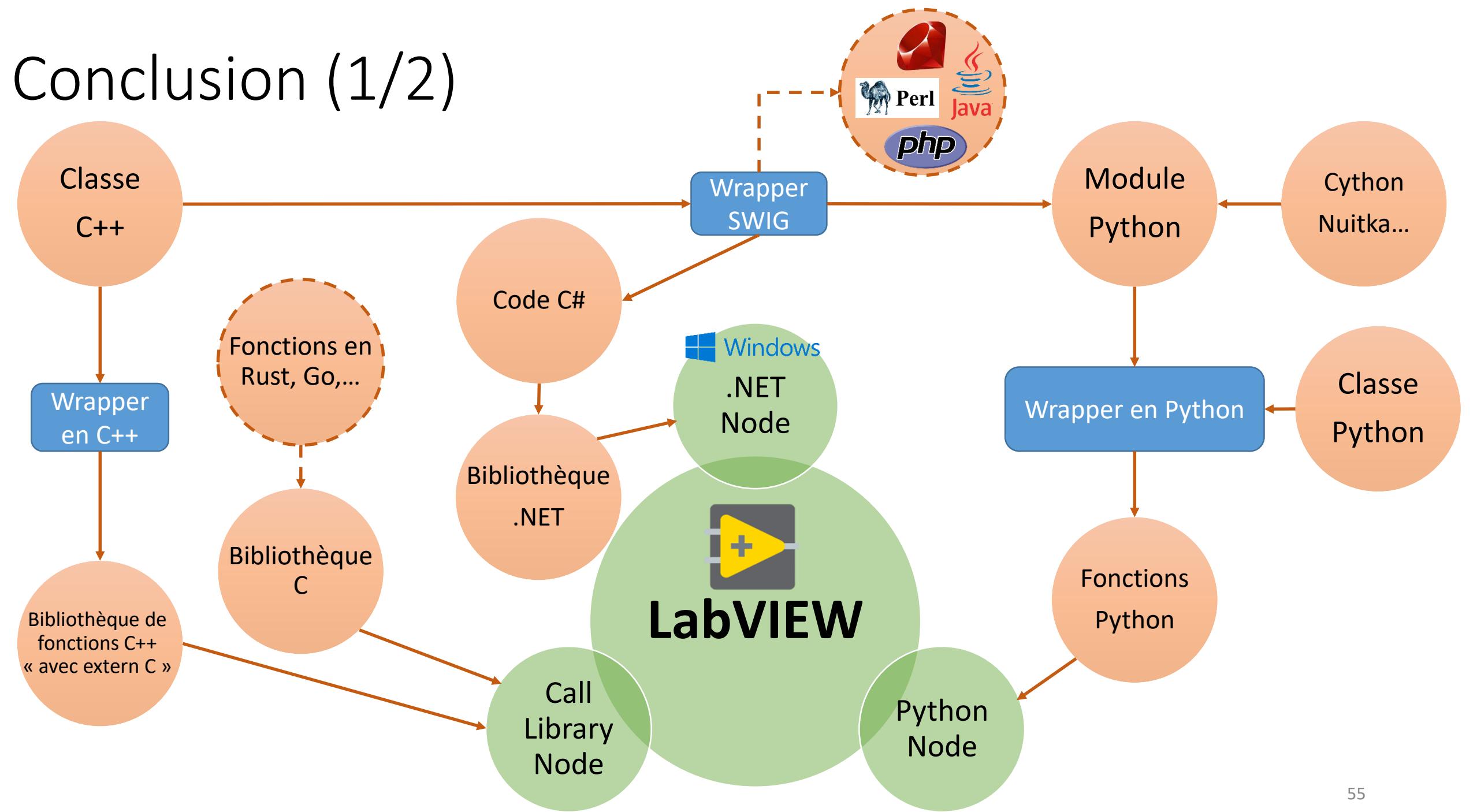
Rq: Sur Linux remplacer « Calc.dll » par « Calc.so »

Objet C++

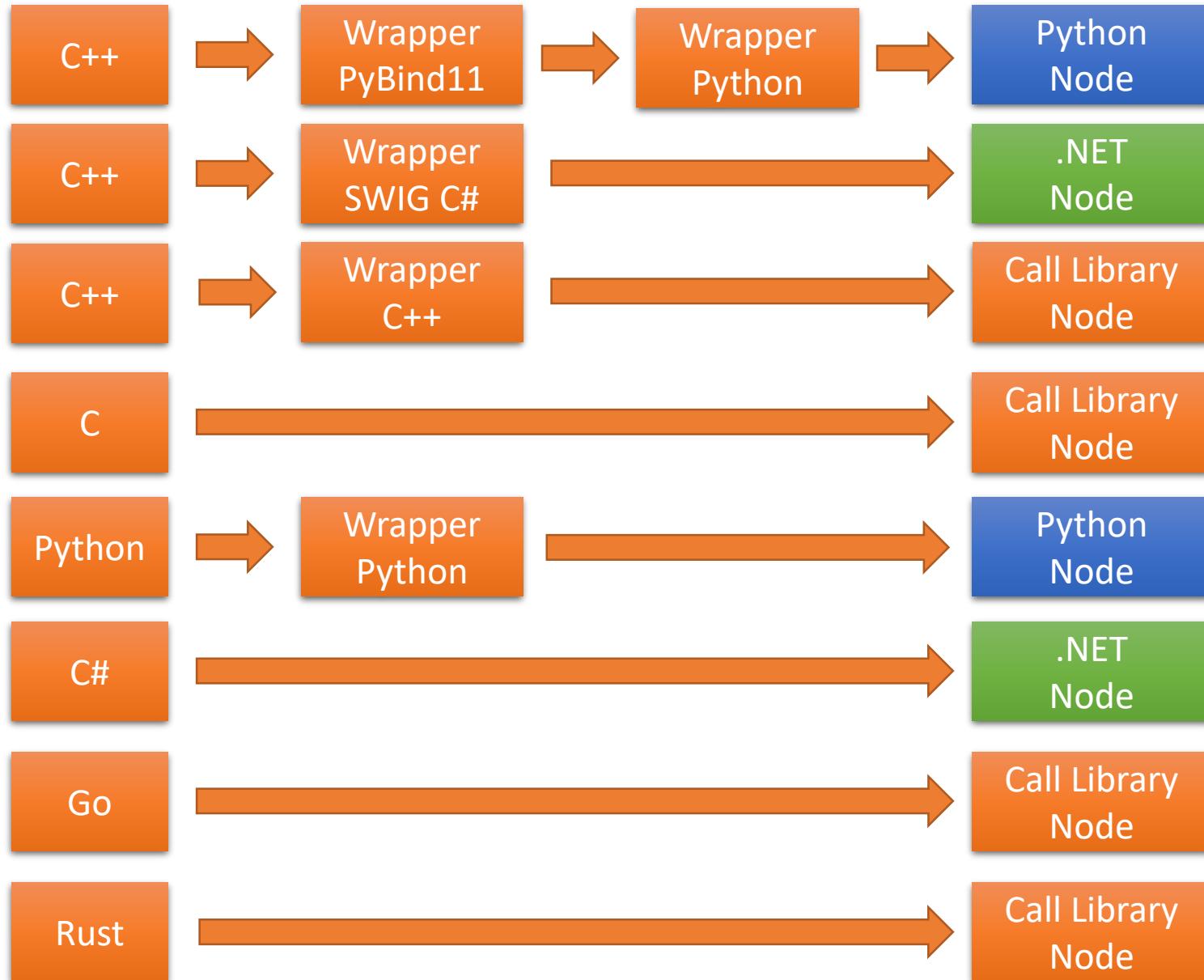
C++ avec un wrapper (3/3)



Conclusion (1/2)



Conclusion (2/2)



Référence

- SWIG: <http://www.swig.org/>
- PyBind11:
 - <https://github.com/pybind/pybind11>
 - <https://pybind11.readthedocs.io/en/master/>
- Wrapper C/C++: <https://nachtimwald.com/2017/08/18/wrapping-c-objects-in-c/>
- Python avec C/C++: <https://docs.python.org/fr/3/extending/building.html>
- Python et ctypes: <https://deptinfo-ensip.univ-poitiers.fr/ENS/doku/doku.php/stu:progc:ctypes>
- Go:
 - <https://medium.com/learning-the-go-programming-language/calling-go-functions-from-other-languages-4c7d8bcc69bf>
 - <https://golang.org/>
- Rust: <https://doc.rust-lang.org/1.2.0/book/rust-inside-other-languages.html>
- Microsoft Visual Studio:
 - <https://docs.microsoft.com/en-us/cpp/build/reference/compiler-options-listed-by-category?view=vs-2019>
 - <https://docs.microsoft.com/fr-fr/cpp/build/walkthrough-compiling-a-native-cpp-program-on-the-command-line?view=vs-2019>
 - <https://docs.microsoft.com/fr-fr/cpp/build/walkthrough-compile-a-c-program-on-the-command-line?view=vs-2019>

Annexe

Installer g++ et gcc sur Windows (1/2)

Aller sur <http://mingw-w64.org/doku.php/download>, télécharger et installer la version MingW-W64-builds

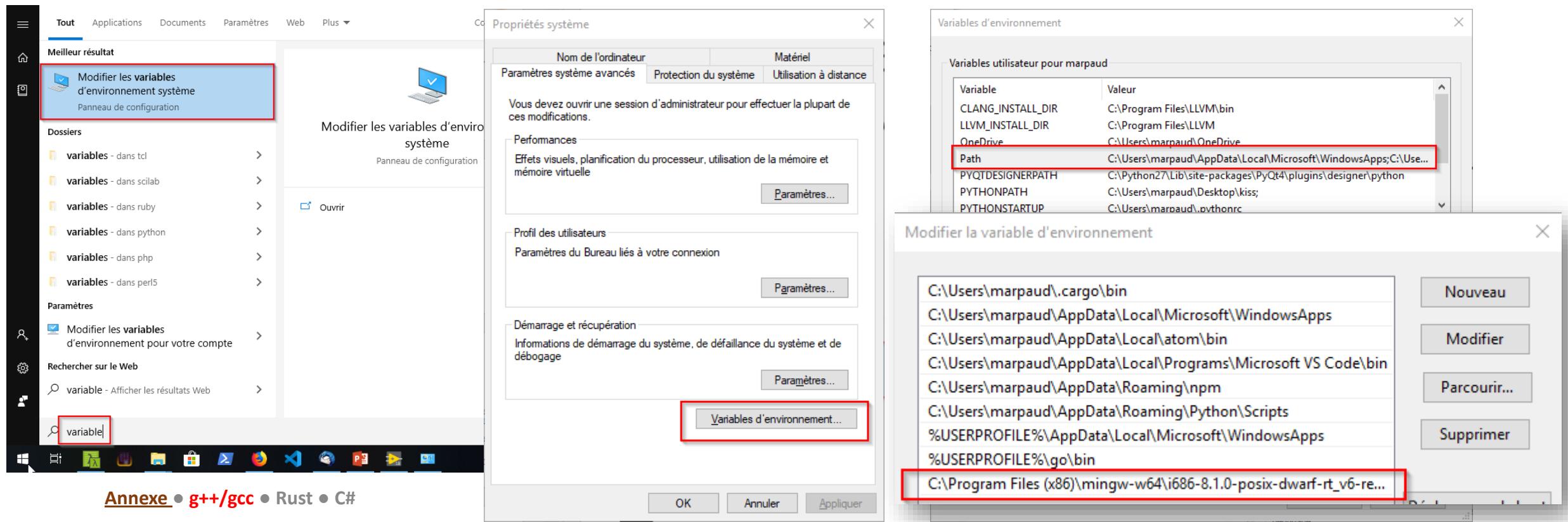
Pre-built toolchains and packages

	Version	Host	GCC / Mingw-w64 Version	Languages	Additional Software in Package Manager
 Arch Linux	Arch Linux		8.2.0/5.0.4	Ada, C, C++, Fortran, Obj-C, Obj-C++	305, full list: Show
 Cygwin	Rolling	Windows 	5.4.0/5.0.2	Ada, C, C++, Fortran, Obj-C	5 (bzip2, libgcrypt, libgpg-error, minizip, xz, zlib)
			4.6.3/2.0.3		
 Debian	Debian 8 (Jessie)		4.9.1/3.2.0	Ada, C, C++, Fortran, Obj-C, Obj-C++, OCaml	2 (gdb, nsis)
	Debian 9 (Stretch)		6.3.0/5.0.0		9 (gdb, libassuan, libgcrypt, libgpg-error, libksba, libnpth, nsis, win-iconv, zlib)
	Debian 10 (Buster)		8.3.0/6.0.0		
 Fedora	Fedora 19		4.8.1/?		149, full list: Show
 MacPorts	Rolling	macOS 	8.2.0/5.0.4	C, C++, Fortran, Obj-C, Obj-C++	1 (nsis)
 MingW-W64-builds	Rolling	Windows 	7.2.0/5.0.3	C, C++, Fortran	4 (gdb, libconf, python, zlib)
 Msys2	Rolling	Windows 	8.2.0/trunk	Ada, C, C++, Fortran, Obj-C, Obj-C++, OCaml	many
			12.04 Precise Pangolin		
			14.04 Trusty Tahr		
 Ubuntu	14.10 Utopic Unicorn		4.9.1/3.1.0		2 (nsis, gdb)
	15.04 Vivid Vervet		4.9.2/3.2.0		
	15.10 Wily Werewolf		4.9.2/4.0.2		
	16.04 Xenial Xerus		5.3.1/4.0.4		3 (nsis, gdb, zlib)
 Win-Builds	1.5	Windows  Linux 	4.8.3/3.3.0	C, C++	91, full list: Show

Annexe

Installer g++ et gcc sur Windows (2/2)

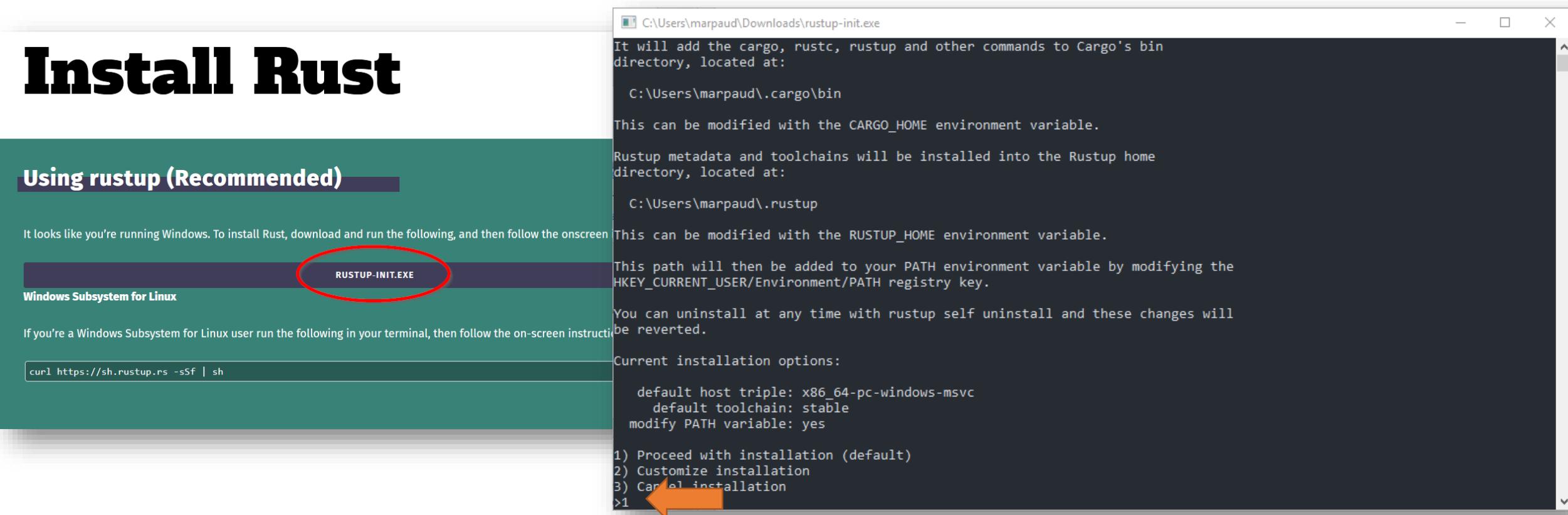
Modifier la variable d'environnement « **Path** » pour pointer vers les dossiers « **bin** » de MinGW: *C:\Program Files (x86)\mingw-w64\i686-8.1.0-posix-dwarf-rt_v6-rev0\mingw32\bin* (par exemple pour moi)



Annexe

Installation de Rust sur Windows (1/1)

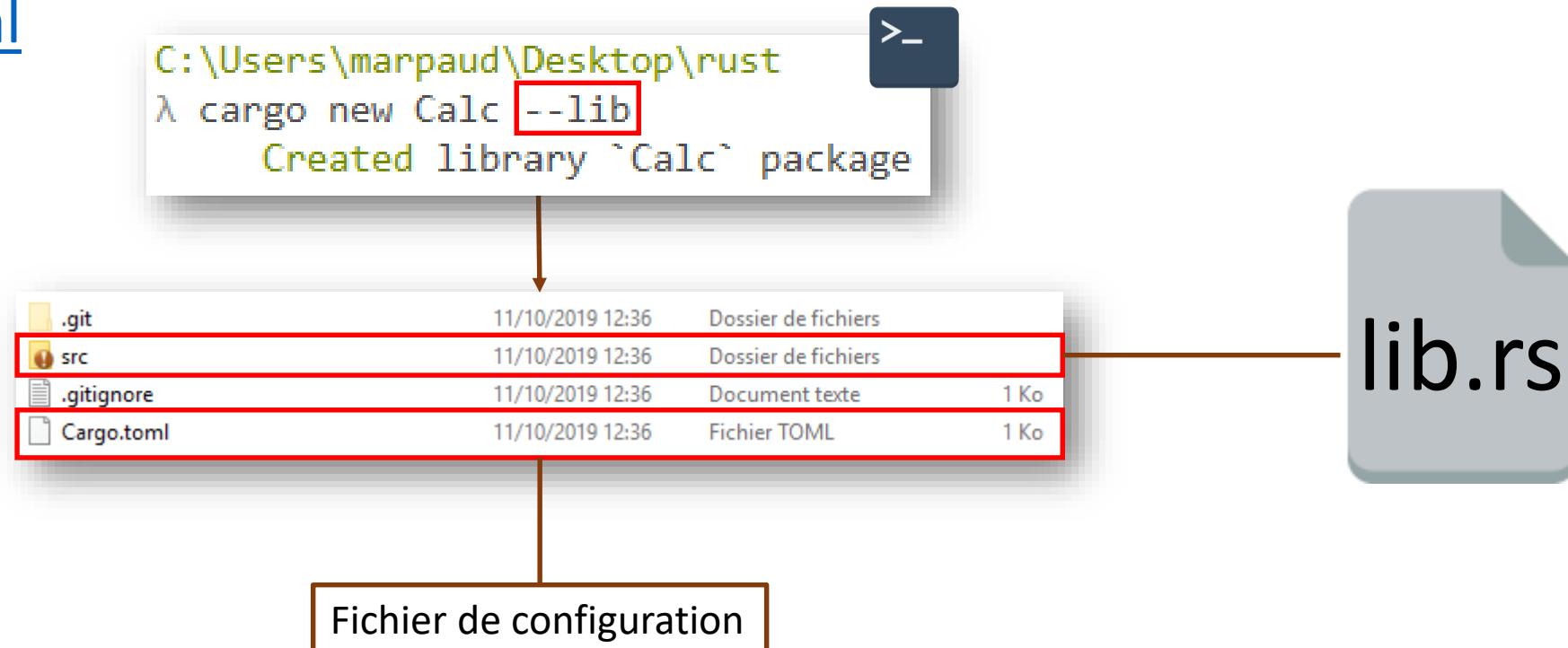
Aller sur <https://www.rust-lang.org/tools/install>



Annexe

Rust, Cargo et librairie (1/1)

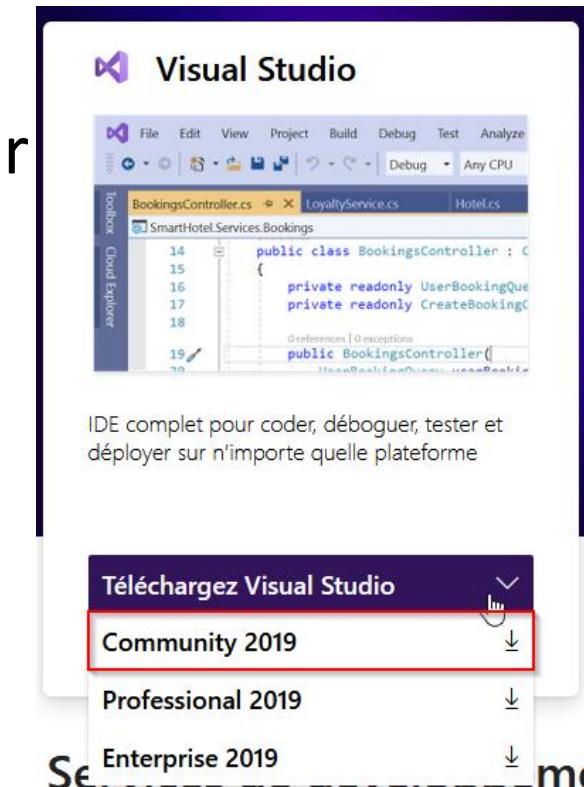
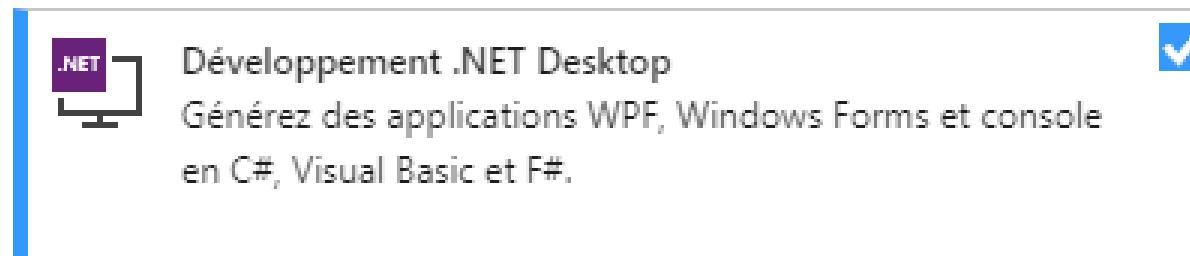
Créer un projet **Cargo** → <https://doc.rust-lang.org/book/ch01-03-hello-cargo.html> et <https://doc.rust-lang.org/cargo/commands/cargo-new.html>



Annexe

C# sur Windows (1/2)

- Pour pouvoir compiler du C#, il faut installer **Visual Studio** :
<https://visualstudio.microsoft.com/fr/> , ici on a utilisé la version « community »
- Une fois installé, ouvrir **Visual Studio Installer**, et installer « Développement .NET Desktop »



Annexe

C# sur Windows (2/2)

Si vous voulez utiliser les lignes de commande pour compiler du C#, il faut chercher les « prompt » dans le menu démarer, et choisir la version qui vous convient (32bits ou 64 bits)

