SWIG Tutos

From C++/C to python

Jean-Charles Lambert

Outline

- Description
- C example
- C++ example
- Advanced example with numpy

Swig tutorial on gitlab.lam.fr

Online tutorial

http://goo.gl/stbpUh

Clone the project

```
git clone https://gitlab.lam.fr/jclamber/swig-tutos.git
```

Or

```
git clone git@gitlab.lam.fr:jclamber/swig-tutos.git
```

Description : what is it ?

- Swig is an interface compiler
- Swig **connects** C++/C language to scripting language :
  - Python, Perl, Ruby or TCL
- Uses an interface file (filename**.i**) to generate wrapper code to build connexions with scripting language
- **Is numpy "aware"** !

Description : why do we need swig ?

- To speed up your pythons programs
  - C/C++ libraries run computing intensive parts of the code
    - Loops
    - MKL / OpenMP
  - Python get results via numpy arrays
- To debug and test your libraries c/c++
  - Python can call every functions of shared library
  - It’s easy to write python scripts

Description : concept

Your library files

myfile.c myfile.h

Description : concept

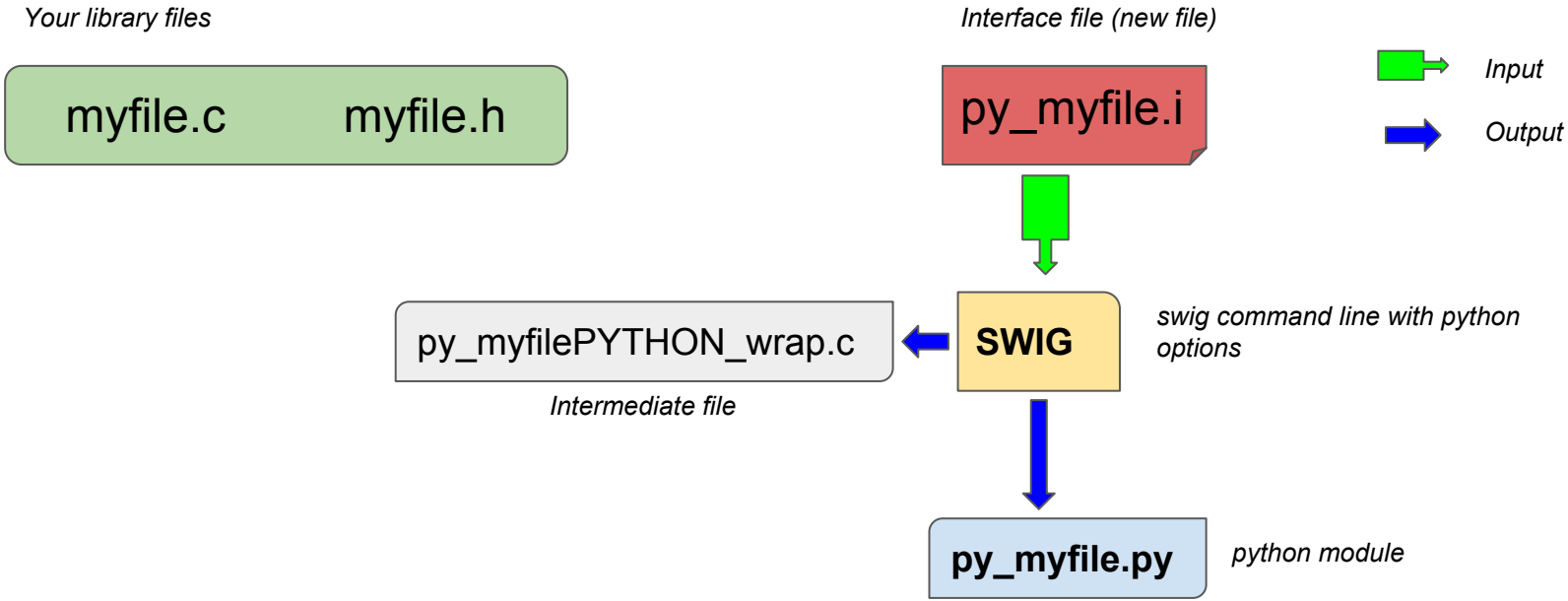
Your library files

myfile.c myfile.h

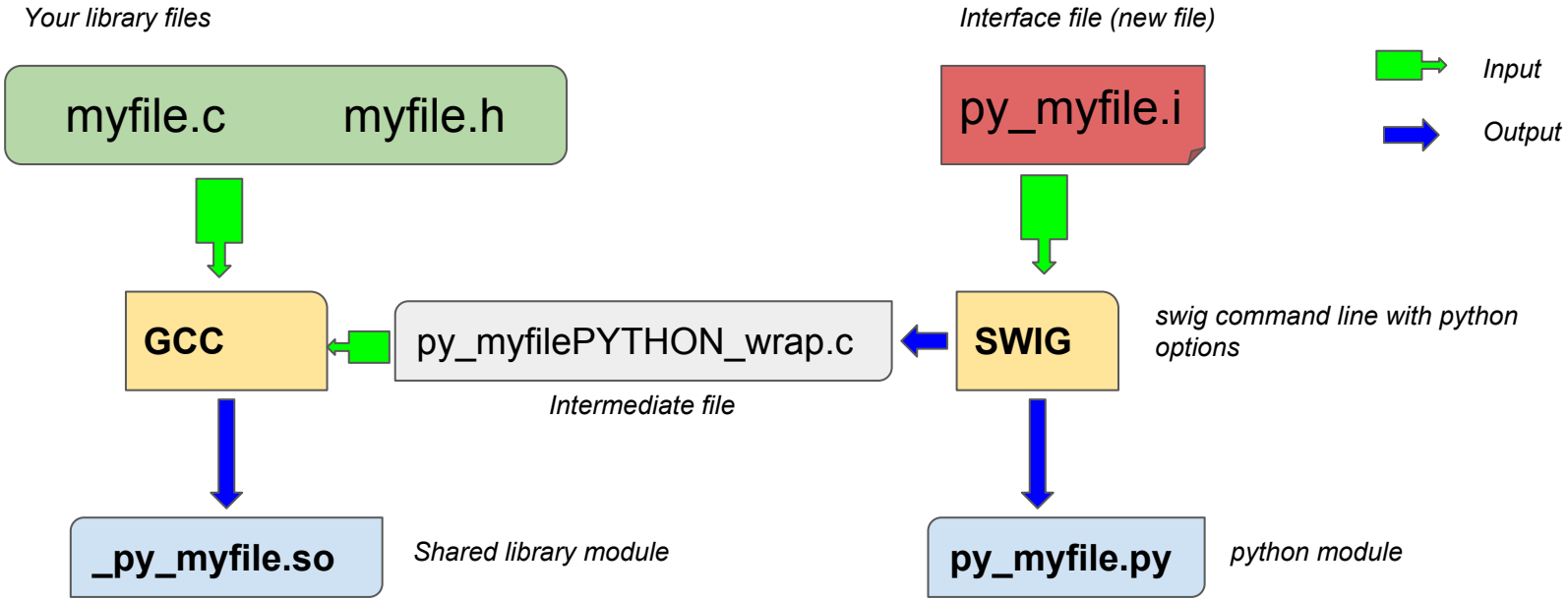
Interface file (new file)

py_myfile.i

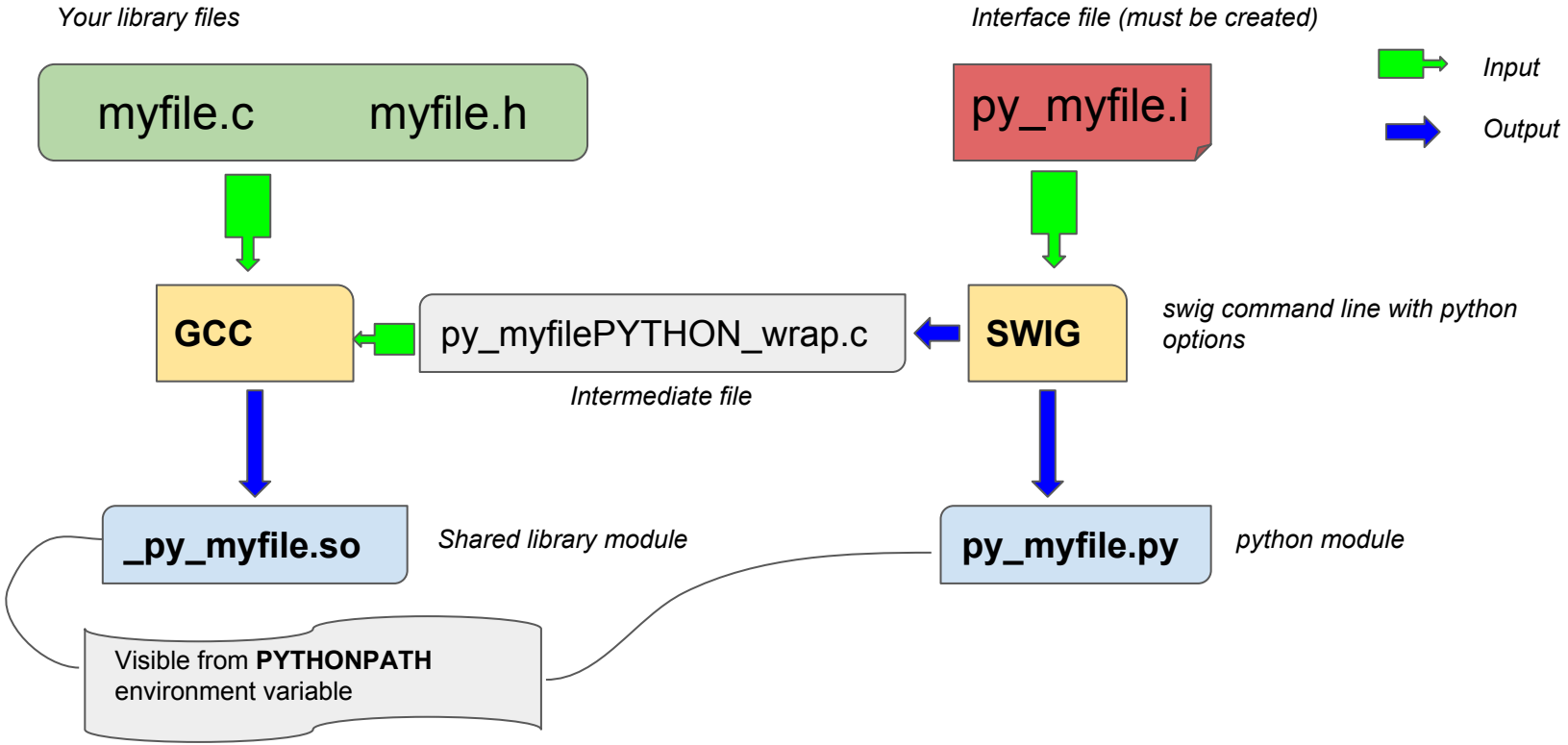
Description : concept



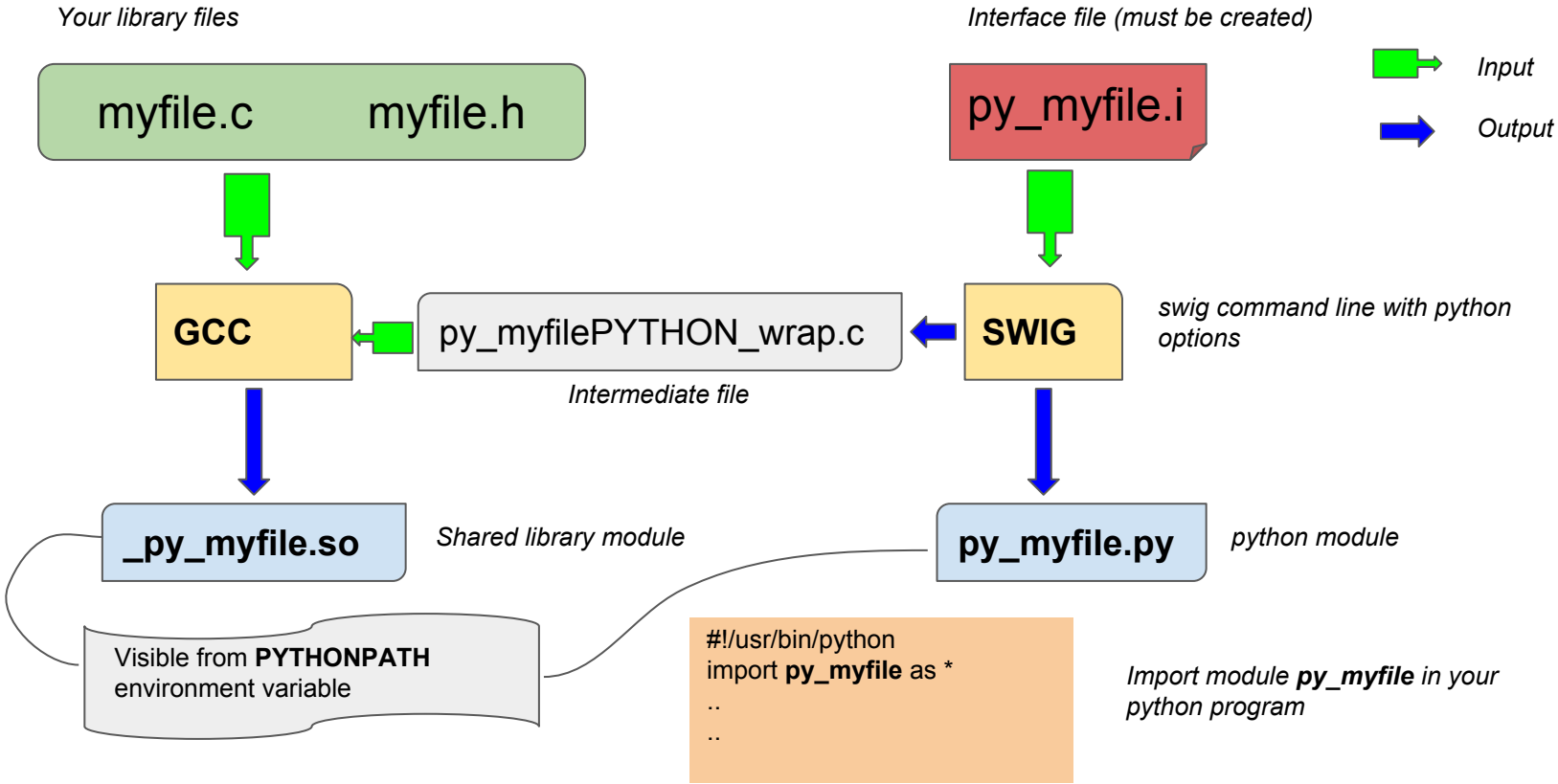
Description : concept



Description : concept



Description : concept



Example : C language (ex01)

mylib.h

```
#ifndef MYLIB_H
#define MYLIB_H
float mult(float a, float b);
void print_msg(char * txt);
#endif
```

mylib.c

```
#include "mylib.h"
#include <stdio.h>
float mult(float a, float b)
{
  return a*b;
}
void print_msg(char * txt)
{
  printf("%s\n",txt);
}
```

Example : C language (ex01)

mylib.h

```
#ifndef MYLIB_H
#define MYLIB_H
float mult(float a, float b);
void print_msg(char * txt);
#endif
```

mylib.c

```
#include "mylib.h"
#include <stdio.h>
float mult(float a, float b)
{
  return a*b;
}
void print_msg(char * txt)
{
  printf("%s\n",txt);
}
```

py_mylib.i (swig interface file)

```
%module py_mylib

%{
  #include "mylib.h"
%}

%include "mylib.h"
```

Example : C language (ex01)

mylib.h

```
#ifndef MYLIB_H
#define MYLIB_H
float mult(float a, float b);
void print_msg(char * txt);
#endif
```

mylib.c

```
#include "mylib.h"
#include <stdio.h>
float mult(float a, float b)
{
  return a*b;
}
void print_msg(char * txt)
{
  printf("%s\n",txt);
}
```

py_mylib.i

```
%module py_mylib

%{
  #include "mylib.h"
%}

%include "mylib.h"
```

- **%module py_mylib** => python module name

Example : C language (ex01)

mylib.h

```
#ifndef MYLIB_H
#define MYLIB_H
float mult(float a, float b);
void print_msg(char * txt);
#endif
```

mylib.c

```
#include "mylib.h"
#include <stdio.h>
float mult(float a, float b)
{
  return a*b;
}
void print_msg(char * txt)
{
  printf("%s\n",txt);
}
```

py_mylib.i

```
%module py_mylib

%{
  #include "mylib.h"
%}

%include "mylib.h"
```

- **%module py_mylib** => python module name
- **%{ … %}** must include header files, **mandatory** for wrapper to compile

Example : C language (ex01)

mylib.h

```
#ifndef MYLIB_H
#define MYLIB_H
float mult(float a, float b);
void print_msg(char * txt);
#endif
```

mylib.c

```
#include "mylib.h"
#include <stdio.h>
float mult(float a, float b)
{
  return a*b;
}
void print_msg(char * txt)
{
  printf("%s\n",txt);
}
```

py_mylib.i

```
%module py_mylib

%{
  #include "mylib.h"
%}

%include "mylib.h"
```

- **%module py_mylib** => python module name
- **%{ … %}** must include header files, **mandatory** for wrapper to compile
- **%include "mylib.h"** parse all functions exported to python

Example : C language (ex01)

mylib.h

```
#ifndef MYLIB_H
#define MYLIB_H
float mult(float a, float b);
void print_msg(char * txt);
#endif
```

mylib.c

```
#include "mylib.h"
#include <stdio.h>
float mult(float a, float b)
{
  return a*b;
}
void print_msg(char * txt)
{
  printf("%s\n",txt);
}
```

py_mylib.i

```
%module py_mylib

%{
  #include "mylib.h"
%}

%include "mylib.h"
```

- **%module py_mylib** => python module name
- **%{ … %}** must include header files, **mandatory** for wrapper to compile
- **%include "mylib.h"** parse all functions exported to python
- Instead, **float mult(float,float)** will only export this function

Example : C language (ex01) - compilation

Swig command line

swig -python **py_mylib.i**

py_mylib.py
py_mylib_wrap.c

Example : C language (ex01) - compilation

Swig command line

```
swig -python py_mylib.i
```

py_mylib.py
py_mylib_wrap.c

Building shared library

```
gcc -shared -fpic py_mylib_wrap.c mylib.c  -I/usr/include/python2.7  -o _py_mylib.so
```

Example : C language (ex01) - compilation

Swig command line

```
swig -python py_mylib.i
```

→ py_mylib.py
py_mylib_wrap.c

Building shared library

```
gcc -shared -fpic py_mylib_wrap.c mylib.c  -I/usr/include/python2.7  -o _py_mylib.so
```

Mandatory: name shared library convention, *_INTERFACENAME*.so (**_py_mylib.so**)

Here *INTERFACENAME* is py_mylib

Example : C language - compilation via setup.py

setup.py

```
from distutils.core import setup, Extension

setup(
    ext_modules = [
        Extension("_py_mylib", sources=["mylib.c", "py_mylib.i"])
    ]
)
```

Command line to build module

```
python setup.py build_ext -i
```

Example : C language (ex01) - test

```
python
>>> import py_mylib
>>> r=py_mylib.mult(2,4)
>>> print r
8.0
>>> py_mylib.print_msg("hello world")
hello world
>>> py_mylib.print_msg("hi there ! "+str(py_mylib.mult(2,4)))
hi there ! 8.0
```

Example : C language (ex01) - test

```
python
>>> import py_mylib
>>> r=py_mylib.mult(2,4)
>>> print r
8.0
>>> py_mylib.print_msg("hello world")
hello world
>>> py_mylib.print_msg("hi there ! "+str(py_mylib.mult(2,4)))
hi there ! 8.0
```

WARNING : do no forget to export **PYTHONPATH** with the location of your module+shared library to use globally your python program

Example : C++ language (ex02) - presentation

my_class.h

```
#ifndef MYCLASS_H
#define MYCLASS_H
class MyClass {
  public:
    MyClass(float _v) { value=_v; };
    float getSqr();
    float getSqr(float _v);
  private:
    float value;
};
#endif
```

my_class.cc

```
#include "my_class.h"

float MyClass::getSqr() {
  return getSqr(value);
}
float MyClass::getSqr(float value) {
  return value*value;
}
```

Example : C++ language (ex02) - presentation

my_class.h

```
#ifndef MYCLASS_H
#define MYCLASS_H
class MyClass {
  public:
    MyClass(float _v) { value=_v; };
    float getSqr();
    float getSqr(float _v);
  private:
    float value;
};
#endif
```

my_class.cc

```
#include "my_class.h"

float MyClass::getSqr() {
  return getSqr(value);
}
float MyClass::getSqr(float value) {
  return value*value;
}
```

py_myclass.i

```
%module py_mylib

%{
  #include "mylib.h"
%}

%include "mylib.h"
```

Example : C++ language (ex02) - compilation

Swig command line (*note -c++ switch*)

swig **-c++** -python **py_myclass.i** 

py_myclass.py
py_myclass_wrap.cxx

Example : C++ language (ex02) - compilation

Swig command line (*note -c++ switch*)

```
swig  -c++ -python py_myclass.i
```

py_myclass.py
py_myclass_wrap.cxx

Building shared library

```
g++ -shared -fPIC py_myclass_wrap.cxx my_class.cc -I/usr/include/python2.7 -o _py_myclass.so  -lstdc++
```

Example : C++ language (ex02) - compilation

Swig command line (*note -c++ switch*)

```
swig  -c++ -python py_myclass.i
```

py_myclass.py
py_myclass_wrap.cxx

Building shared library

```
g++ -shared -fPIC py_myclass_wrap.cxx my_class.cc -I/usr/include/python2.7 -o _py_myclass.so  -lstdc++
```

Mandatory: name shared library convention, *_INTERFACENAME*.so (**_py_myclass.so)**

Here *INTERFACENAME* is py_myclass

Example : C++ language - compilation via setup.py

setup.py

```
from distutils.core import setup, Extension

setup(
    ext_modules = [
        Extension("_py_myclass", sources=["my_class.cc", "py_myclass.i"],
        swig_opts=['-c++'])
    ]
)
```

Command line to build module

```
python setup.py build_ext -i
```

Example : C++ language (ex02) - test

```
python
>>> from py_myclass import *
>>> x=MyClass(2)
>>> x.getSqr()
4.0
>>> x.getSqr(10)
100.0
```

WARNING : do not forget to export **PYTHONPATH** with the location of your module+shared library to use globally your python program

How to deal with numpy array and swig ?

C routine to re-implement range function:

➢ void range(double* **rangevec**, int n);

How to deal with numpy array and swig ?

C routine to re-implement range function:

➢ void range(double* **rangevec**, int n);

What is **rangevec** variable for python ?

- A single value passed by address ?
- An array ? If so, which length ?
- Is it input only ? Output only ? Input-Output ?

Swig needs a special interface file to make data types conversions => **numpy.i**

numpy.i : ARGOUT_ARRAY1 (exo 3) - presentation

From numpy point of view :

ARGOUT_ARRAY are arrays return from a function

Example : numpy.arange(value) returns an ARGOUT_ARRAY

```
python
>>> import numpy as np
>>> x=np.arange(10)
```

numpy.i : ARGOUT_ARRAY1 (exo 3) - files

my_range.h

```
void range(int *rangevec, int n);
```

my_range.c

```
void range(int *rangevec, int n)
{
    int i;
    for (i=0; i< n; i++)
        rangevec[i] = i;
}
```

numpy.i : ARGOUT_ARRAY1 (exo 3) - files

my_range.h

```
void range(int *rangevec, int n);
```

my_range.c

```
void range(int *rangevec, int n)
{
    int i;
    for (i=0; i< n; i++)
        rangevec[i] = i;
}
```

py_myrange.i

```
%module py_myrange

%{
  #define SWIG_FILE_WITH_INIT
  #include "my_range.h"
%}

%include "numpy.i"

%init %{
    import_array();
%}

%apply (int* ARGOUT_ARRAY1, int DIM1) {(int* rangevec, int n)}

%include "my_range.h"
```

numpy.i : ARGOUT_ARRAY1 (exo 3) - files

my_range.h

```
void range(int *rangevec, int n);
```

Same syntax !!

my_range.c

```
void range(int *rangevec, int n)
{
    int i;
    for (i=0; i< n; i++)
        rangevec[i] = i;
}
```

py_myrange.i

```
%module py_myrange

%{
  #define SWIG_FILE_WITH_INIT
  #include "my_range.h"
%}

%include "numpy.i"

%init %{
    import_array();
%}

%apply (int* ARGOUT_ARRAY1, int DIM1) {(int* rangevec, int n)}

%include "my_range.h"
```

Same syntax !!

In header file and in interface file
It's mandatory to have same syntax for arguments which are type mapped !!

numpy.i : ARGOUT_ARRAY1 (exo 3) - compilation

setup.py

```
from distutils.core import setup, Extension
import numpy
#  find out numpy include directory.
try:
    numpy_include = numpy.get_include()
except AttributeError:
    numpy_include = numpy.get_numpy_include()

setup(
    ext_modules = [
        Extension("_py_myrange", sources=["my_range.c", "py_myrange.i"],
        include_dirs = [numpy_include])
    ]
)
```

Command line to build module

```
python setup.py build_ext -i
```

numpy.i : ARGOUT_ARRAY1 (exo 3) - test

ARGOUT_ARRAY are output numpy array, we just give the array dimension as input

```
python
>>> import py_myrange
>>> py_myrange.range(5)
array([0, 1, 2, 3, 4], dtype=int32)
>>> a=py_myrange.range(5)
>>> a
array([0, 1, 2, 3, 4], dtype=int32)
>>>
```

numpy.i : INPLACE_ARRAY1 (exo 4) - presentation

From numpy point of view :

INPLACE_ARRAY means that array **is passed** to the function AND **might be changed** in the function

numpy.i : INPLACE_ARRAY1 (exo 4) - files

my_inplace.h

```
void inplace(double *invec, int n);
```

my_inplace.c

```
void inplace(double *invec, int n)
{
    int i;
    for (i=0; i< n; i++)
        invec[i] = 2*invec[i];
}
```

INPLACE_ARRAY means that array **is passed** to the function AND **might be changed** in the function

numpy.i : INPLACE_ARRAY1 (exo 4) - files

my_inplace.h

```
void inplace(double *invec, int n);
```

my_inplace.c

```
void inplace(double *invec, int n)
{
    int i;
    for (i=0; i< n; i++)
        invec[i] = 2*invec[i];
}
```

py_myinplace.i

```
%module py_myinplace

%{
  #define SWIG_FILE_WITH_INIT
  #include "my_inplace.h"
%}

%include "numpy.i"

%init %{
    import_array();
%}

%apply (double* INPLACE_ARRAY1, int DIM1) {(double* invec, int n)}

%include "my_inplace.h"
```

INPLACE_ARRAY means that array **is passed** to the function AND **might be changed** in the function

numpy.i : INPLACE_ARRAY1 (exo 4) - files

my_inplace.h

```
void inplace(double *invec, int n);
```

Same syntax !!

my_inplace.c

```
void inplace(double *invec, int n)
{
    int i;
    for (i=0; i< n; i++)
        invec[i] = 2*invec[i];
}
```

py_myinplace.i

```
%module py_myinplace

%{
  #define SWIG_FILE_WITH_INIT
  #include "my_inplace.h"
%}

%include "numpy.i"

%init %{
    import_array();
%}

%apply (double* INPLACE_ARRAY1, int DIM1) {(double* invec, int n)}

%include "my_inplace.h"
```

Same syntax !!

INPLACE_ARRAY means that array **is passed** to the function AND **might be changed** in the function

numpy.i : INPLACE_ARRAY1 (exo 4) - compilation

setup.py

```
from distutils.core import setup, Extension
import numpy
#  find out numpy include directory.
try:
    numpy_include = numpy.get_include()
except AttributeError:
    numpy_include = numpy.get_numpy_include()

setup(
    ext_modules = [
        Extension("_py_myinplace", sources=["my_inplace.c", "py_myinplace.i"],
        include_dirs = [numpy_include])
    ]
)
```

Command line to build module

```
python setup.py build_ext -i
```

numpy.i : INPLACE_ARRAY1 (exo 4) - test

```
python
>>> import numpy
>>> import py_myinplace
>>> a = numpy.array([1,2,3],'d')
>>> py_myinplace.inplace(a)
>>> a
array([ 2.,  4.,  6.])
```

numpy.i : ARGOUTVIEW_ARRAY1 (exo 5) - presentation

From numpy point of view :

ARGOUTVIEW_ARRAY are arrays dynamically allocated by the C/C++ library and return by a python function as numpy array.

It’s dangerous because python cannot manage memory de-allocation of these arrays.

numpy.i : ARGOUTVIEW_ARRAY1 (exo 5) - files

my_outview.h

```
#include <stdlib.h>
#include <stdio.h>
void readFile(char * f);
void getData(double ** array, int * n);
```

my_outview.c

```
#include "my_outview.h"

static double * my_array=NULL;
static int nelem;

void readFile(char * f) {
  int i,status;
  FILE * in = fopen(f,"rt");
  status=fscanf(in,"%d",&nelem);
  if (my_array) free((double *) my_array);
  my_array = (double *) malloc(sizeof(double)*nelem);
  for (i=0; i<nelem; i++) status=fscanf(in,"%lf",my_array+i);
}

void getData(double ** array, int * n) {
  int i;
  *n=nelem;
  *array=my_array;
}
```

numpy.i : ARGOUTVIEW_ARRAY1 (exo 5) - files

my_outview.h

```
#include <stdlib.h>
#include <stdio.h>
void readFile(char * f);
void getData(double ** array, int * n);
```

my_outview.c

```
#include "my_outview.h"

static double * my_array=NULL;
static int nelem;

void readFile(char * f) {
  int i,status;
  FILE * in = fopen(f,"rt");
  status=fscanf(in,"%d",&nelem);
  if (my_array) free((double *) my_array);
  my_array = (double *) malloc(sizeof(double)*nelem);
  for (i=0; i<nelem; i++) status=fscanf(in,"%lf",my_array+i);
}

void getData(double ** array, int * n) {
  int i;
  *n=nelem;
  *array=my_array;
}
```

py_myoutview.i

```
%module py_myrange

%{
  #define SWIG_FILE_WITH_INIT
  #include "my_range.h"
%}

%include "numpy.i"

%init %{
    import_array();
%}

%apply (double ** ARGOUTVIEW_ARRAY1, int * DIM1) {( double ** array, int * n)}
%include "my_range.h"
```

numpy.i : ARGOUTVIEW_ARRAY1 (exo 5) - files

my_outview.h

```
#include <stdlib.h>
#include <stdio.h>
void readFile(char * f);
void getData(double ** array, int * n);
```

Same syntax !!

my_outview.c

```
#include "my_outview.h"

static double * my_array=NULL;
static int nelem;

void readFile(char * f) {
  int i,status;
  FILE * in = fopen(f,"rt");
  status=fscanf(in,"%d",&nelem);
  if (my_array) free((double *) my_array);
  my_array = (double *) malloc(sizeof(double)*nelem);
  for (i=0; i<nelem; i++) status=fscanf(in,"%lf",my_array+i);
}

void getData(double ** array, int * n) {
  int i;
  *n=nelem;
  *array=my_array;
}
```

py_myoutview.i

```
%module py_myrange

%{
  #define SWIG_FILE_WITH_INIT
  #include "my_range.h"
%}

%include "numpy.i"

%init %{
    import_array();
%}

%apply (double ** ARGOUTVIEW_ARRAY1, int * DIM1) {( double ** array, int * n)}
%include "my_range.h"
```

Same syntax !!

numpy.i : ARGOUTVIEW_ARRAY1 (exo 5) - compilation

setup.py

```
from distutils.core import setup, Extension
import numpy
#  find out numpy include directory.
try:
    numpy_include = numpy.get_include()
except AttributeError:
    numpy_include = numpy.get_numpy_include()

setup(
    ext_modules = [
        Extension("_py_myoutview", sources=["my_outview.c", "py_myoutview.i"],
        include_dirs = [numpy_include])
    ]
)
```

Command line to build module

```
python setup.py build_ext -i
```

numpy.i : ARGOUTVIEW_ARRAY1 (exo 5) - test

```
python
>>> import py_myoutview
>>> py_myoutview.readFile("data.txt")
>>> a=py_myoutview.getData()
>>> a
array([ 100.,  200.,  300.,  400.,  500.])
>>> b=py_myoutview.getData()
>>> b
array([ 100.,  200.,  300.,  400.,  500.])
>>> b[0]=666.
>>> a
array([ 666.,  200.,  300.,  400.,  500.])
```

data.txt

```
5
100.0
200.0
300.0
400.0
500.0
```

Swig resources on the web

- http://www.swig.org/Doc1.3/Python.html
- https://scipy.github.io/old-wiki/pages/Cookbook/SWIG_NumPy_examples.html
- http://docs.scipy.org/doc/numpy/reference/swig.interface-file.html
- http://web.mit.edu/6.863/spring2011/packages/numpy_src/doc/swig/doc/numpy_swig.html

Swig tutorial on gitlab.lam.fr

Online tutorial

http://goo.gl/stbpUh

Clone the project

```
git clone https://gitlab.lam.fr/jclamber/swig-tutos.git
```

Or

```
git clone git@gitlab.lam.fr:jclamber/swig-tutos.git
```