Python Bindings

We saw that interfacing python with compiled code can provide huge performance gains. There are two main approaches to achieve this:

- Just in time (JIT) compilers: compile and run a python code in real time
  - Numba: jit compiler supporting numpy code
- Ahead of time (AOT) compilers: creation of a compiled library in your machine (this would provide what is called a *binding*)
  - Numba: also provides some AOT functionality (to be re-implemented)
  - Cython: compile a python-like C code or a pure C library
  - f2py: tool part of numpy project allowing to compile and wrap Fortran code
  - pybind11: library to expose C++ types into Python for the creation of C++ bindings
  - Boost.Python: C++ library for Python interoperability

Cython: Fibonacci example

The Fibonacci series is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2} \tag{1}$$

starting with $F_0 = 0$ and $F_1 = 1$.
A basic pure python implementaion[1]:

```
def fibonacci(num):
    fn = 0
    fn1 = 1
    while num-1:
        fn, fn1 = fn1, fn + fn1
        num -= 1
    return fn1


if __name__ == "__main__":

    print(fibonacci(15))
```

[1] code on `python4hpc/examples/compiling/fibo-python/fibonacci.py`

Cython: Fibonacci example

You must annotate your code using a new syntax in between python and C.
Example `fibonacci` function in cython[2]

```
def fibonacci(int num):
    cdef int fn
    cdef int fn1
    fn = 0
    fn1 = 1
    while num-1:
        fn, fn1 = fn1, fn + fn1
        num -= 1
    return fn1
```

To build it is required a sort of makefile, typically called `setup.py`

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize

setup(ext_modules = cythonize("fibolib.pyx"))
```

[2] code files on `python4hpc/examples/compiling/fibo-cython`

Cython: Fibonacci example

The build will produce a binary `.so` object for the library

```
$ python setup.py build_ext --inplace
```

Having this lib on the same directory, it can be imported as a module on a pure python code

```
from fibolib import fibonacci

print(fibonacci(15))
```

Cython: C library

- Cython allows also to wrap C libraries to provide bindings for Python
- Check the example in `python4hpc/examples/compiling/fibo-wrap-c` to see how wrapping works for a C function providing the n_{th} Fibonacci number.
- Steps for building and running the example:

```
$ make
$ python fibonacci.py
The 15th Fibonacci number is: 610
```

f2py: Fortran library

- To wrap Fortran code the `f2py` tool from numpy provides a straighforward approach[3]

```
function fibonacci(n)
    implicit none
    integer, intent(in) :: n
    integer :: fibonacci, fseries(0:n), i
    fseries(0) = 0
    fseries(1) = 1

    do i = 2, n
      fseries(i) = fseries(i - 1) + fseries(i - 2)
    end do
    fibonacci = fseries(n)
end function fibonacci
```

```
import fibolib

print(fibolib.fibonacci(15))
```

```
$ f2py -c -m fibolib fibolib.f90
```

```
$ python fibonacci.py
  610
```

[3] code files on `python4hpc/examples/compiling/fibo-fortran`

Further references and training on the topic

- High Performance Python - 2nd Ed by By Micha Gorelick and Ian Ozsvald
- CSC Python Tutorial: Python in High Performance Computing