

# Incorporating Other Languages into Python

Joey Bernard

# Introduction

- ▶ Python has become the default glue language for science
- ▶ It is not ideal for all cases
- ▶ We will look at how to offload issues to another language

# Installation

- ▶ We need several tools
- ▶ Everything we will discuss involves C/C++
- ▶ You will need Python plus a C/C+ compiler
- ▶ All of this work should be done in a virtual environment (now necessary under Ubuntu)

## Pre-existing Examples

- ▶ Several of the high performance libraries already do this
- ▶ numpy uses C, C++ and FORTRAN (in order of usage)
- ▶ scipy uses C, FORTRAN and C++ (in order of usage)

# Why do this

- ▶ Python is an object oriented language, without static typing
- ▶ This means loops can be horrendous
- ▶ Also have the GIL, throttling multi-process work

# Virtual Environments

- ▶ The first step is creating a virtual environment

```
python -m venv python_project1
```

- ▶ This creates a new directory for your project
- ▶ You can activate it with

```
cd ./python_project1  
. ./bin/activate
```

- ▶ When you are done, you can simply run the command

```
deactivate
```

## First step - Numba

- ▶ In some cases, you just need a slightly faster Python
- ▶ Whenever you try to optimize, remember the quote - ***Early optimization is the root of all evil***
- ▶ You want to do the bare minimum to get the results that you actually need
- ▶ Numba allows for compiling portions of your Python code

## Numba - cont'd

- ▶ Numba is installed using the command

```
pip install numba
```

- ▶ This will install the numba module, along with llvmlite
- ▶ This why you should use virtual environments - to keep your projects clean and isolated



## Numba - cont'd

- ▶ Numba uses decorators to encapsulate your code
- ▶ The most common decorator is `@jit`
- ▶ This decorator has loads of options, including whether to parallelize or whether to target a GPU

# Numba - options

- ▶ ***nogil*** - whether to release the GIL when entering the compiled code
- ▶ ***cache*** - whether to save off compiled code into a file cache to avoid the compiling step each time
- ▶ ***parallel*** - whether to parallelize compiled code when possible (e.g. loops)
- ▶ ***fastmath*** - whether to use strict IEEE 754 math (similar to the GCC flag)

# Numba - explicit typing

- ▶ One issue with Python is that variables are untyped
- ▶ You can assign a type signature as part of the jit decorator
- ▶ For example

```
from numba import jit

@jit(int32(int32,int32))
def my_func(val1, val2):
    return val1 + val2
```

- ▶ This allows numba to know what the data types are and to compile away the usual checks that Python has to do

# Numba - usage

- ▶ Compiling your code is as easy as

```
numba my_code.py
```

- ▶ You can also output debugging information with options like

```
numba my_code.py --annotate
```

OR

```
numba my_code.py --dump_llvm
```

## Next step - Cython

- ▶ Cython allows for adding C/C++ data types, and outputting compiled code
- ▶ You need to annotate your code in order to tell Cython what is expected
- ▶ You will need to have your own C/C++ compiler - ideally the same as the compiler used for Python
- ▶ This becomes easy to mess up under Windows - consider strongly using WSL

# Cython - different notation

## Pure Python

```
def primes(nb_primes: cython.int):
    i: cython.int
    p: cython.int[1000]

    if nb_primes > 1000:
        nb_primes = 1000
    # Only if regular Python
    if not cython.compiled:
        # Make p work almost like
        p = [0] * 1000

    len_p: cython.int = 0
    n: cython.int = 2
    while len_p < nb_primes:
        # Is n prime?
        for i in p[:len_p]:
            if n % i == 0:
```

## Older Cython

```
def primes(int nb_primes):
    cdef int n, i, len_p
    cdef int[1000] p

    if nb_primes > 1000:
        nb_primes = 1000

    # The current number of elements
    len_p = 0
    n = 2
    while len_p < nb_primes:
        # Is n prime?
        for i in p[:len_p]:
            if n % i == 0:
                break
```

# Cython - usage

- ▶ The easiest way to build Cython code is to use setuptools

```
pip install cython  
pip install setuptools
```

# Boost-y binding 1 - Nanobind



## Boost-y binding 2 - pybind11





swig - not just for Python

# pyO3 - a rust option