## **Understanding Numba**

#### Valentin Haenel

Software Engineer - Open Source / Team-Lead Berlin https://anaconda.com

Osteguna 05 Iraila 2019

Version: c6ffd8e28c https://github.com/esc/numba-talk

① ② This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 License.

### Outline

- Introduction
- 2 Going Deeper
- NumPy Support
- 4 Tips and Tricks
- 5 Summary?

### Numba in a Nutshell

- A compiler that might make your code faster
- Requires importing a decorator
- And decorating functions with it
- Numba = NumPy + Mamba (fast snake)

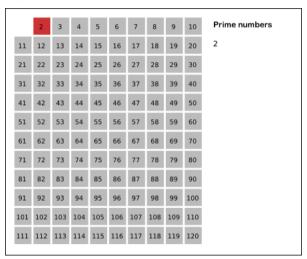
# Numba Explained

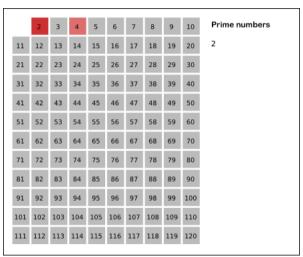
- Numba is a
  - just-in-time
  - type-specializing
  - function compiler
  - for accelerating numerically-focused Python

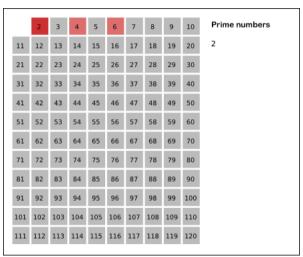
### **LLVM**

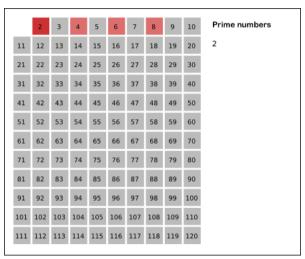
- LLVM is a compiler toolkit
- Numba uses it as a compiler backend
- Access via llvmlite

	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	

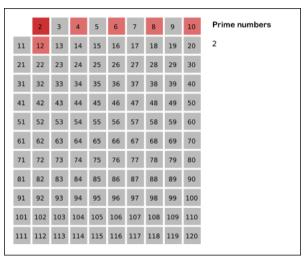


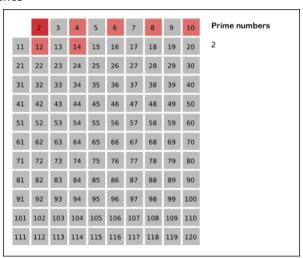


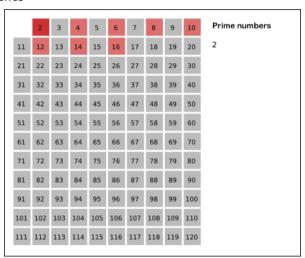


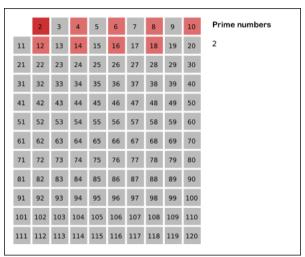


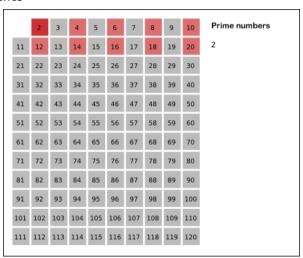
	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	2
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	

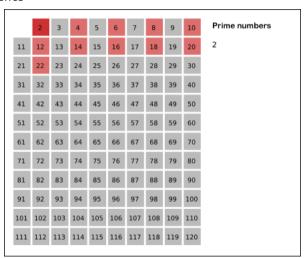


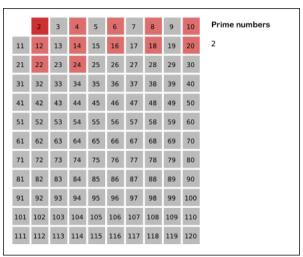


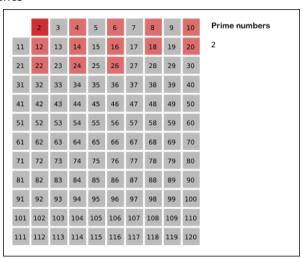


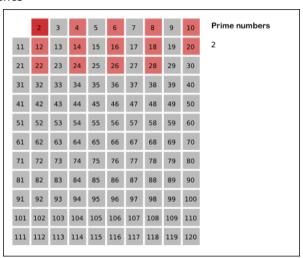


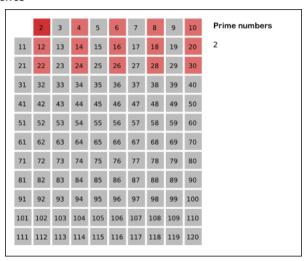


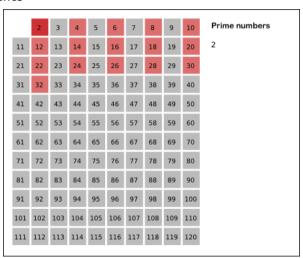


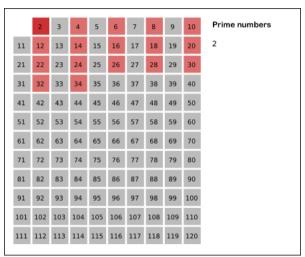


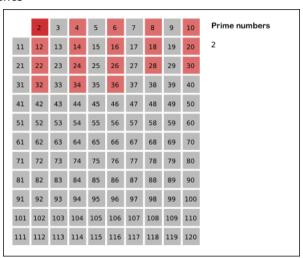


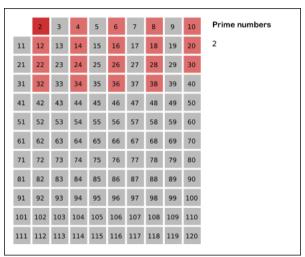


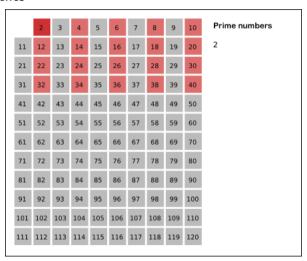


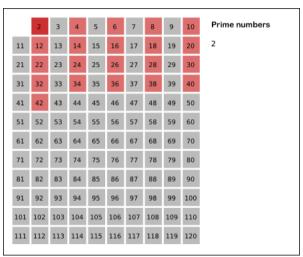


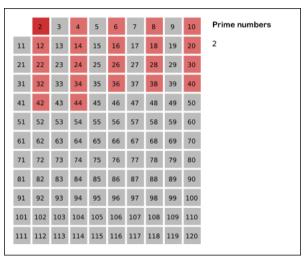


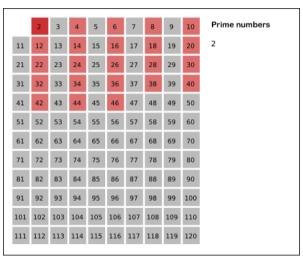


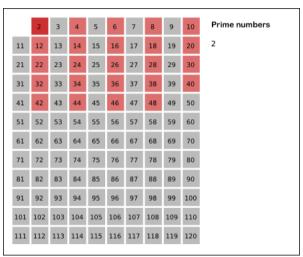


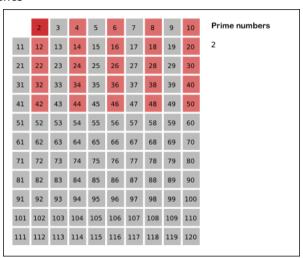


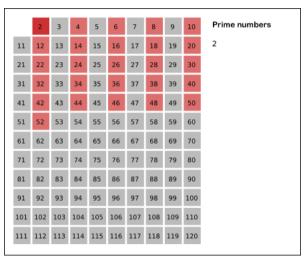


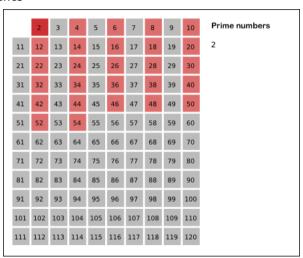


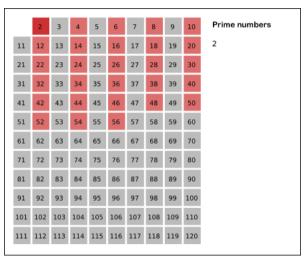


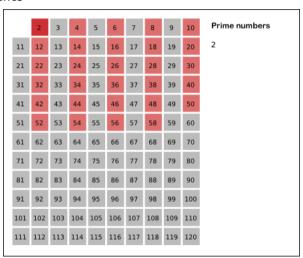


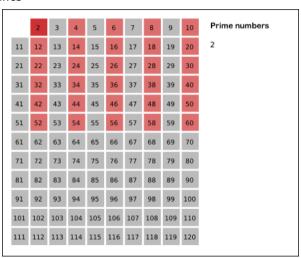


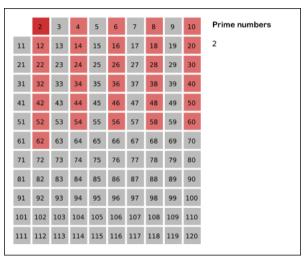




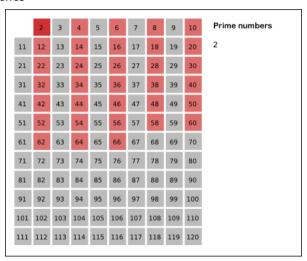


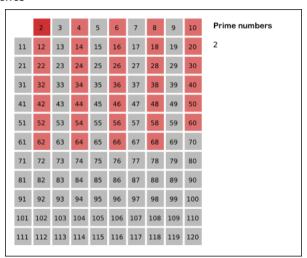


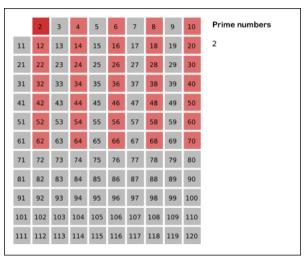


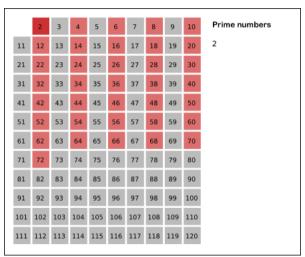


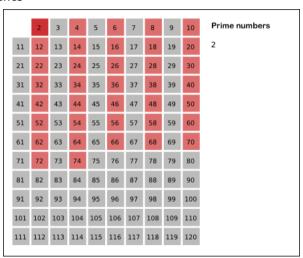


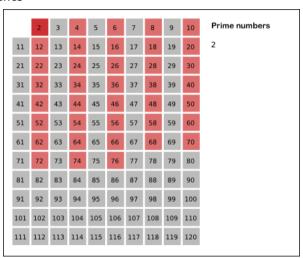


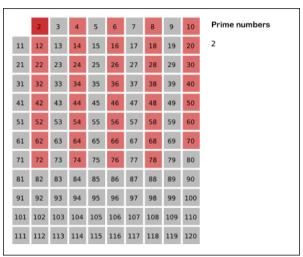


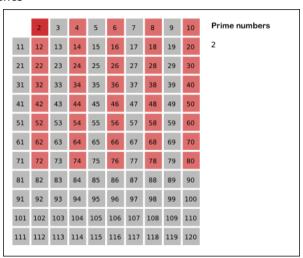


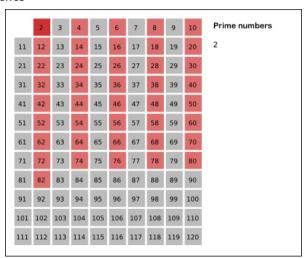


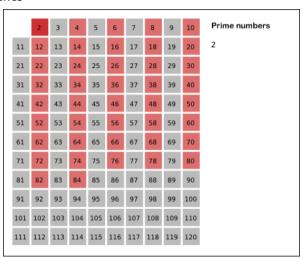


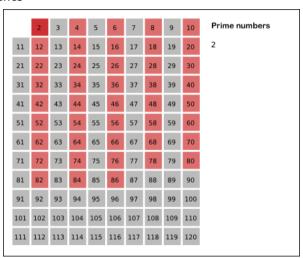


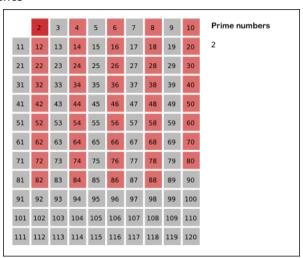


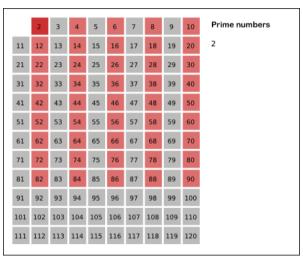


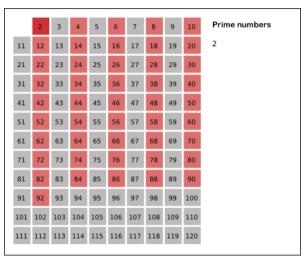


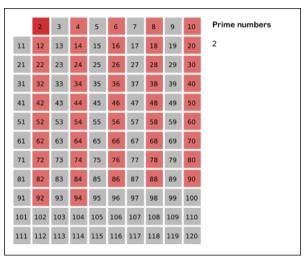


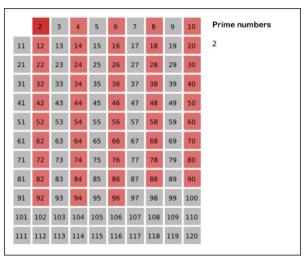


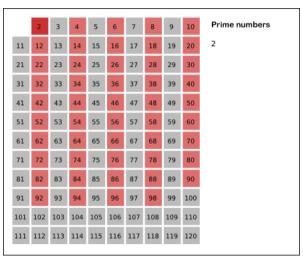


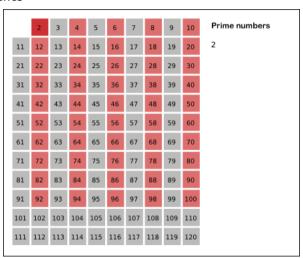


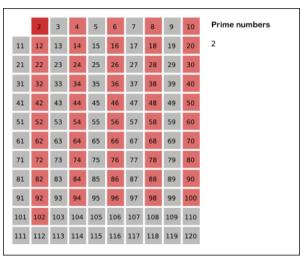


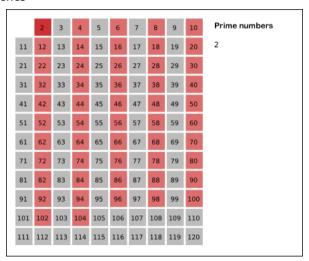


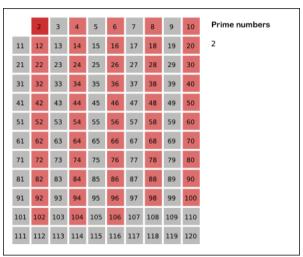


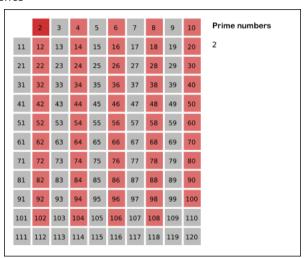


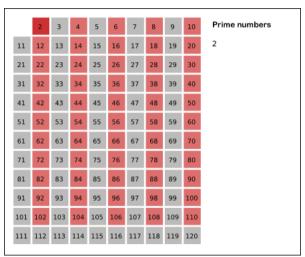


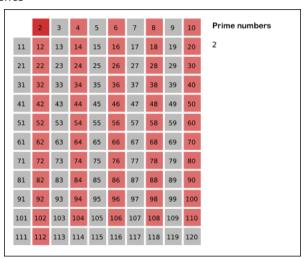


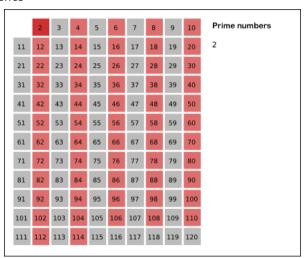


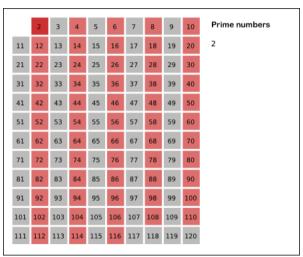


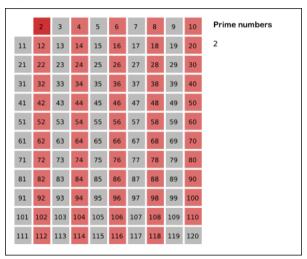


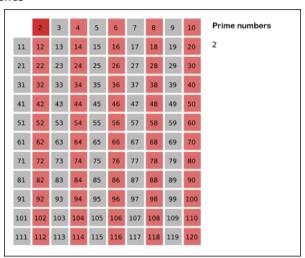


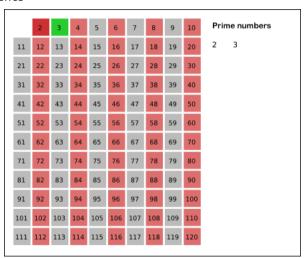


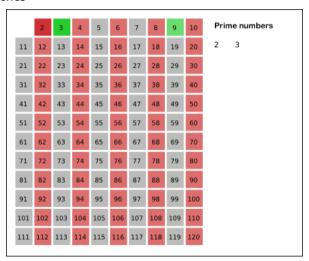


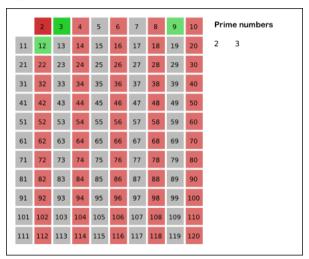


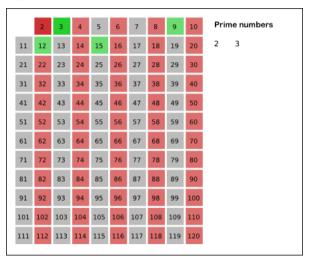


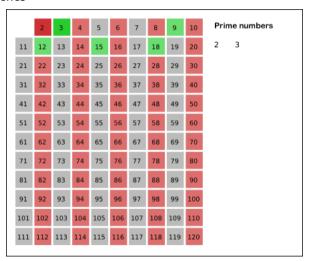


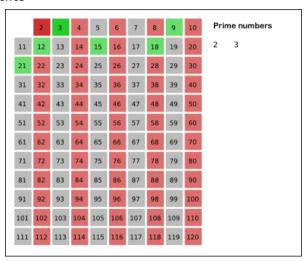


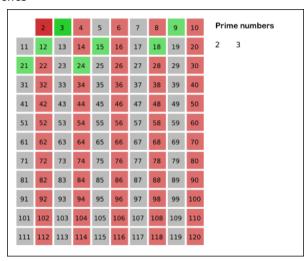


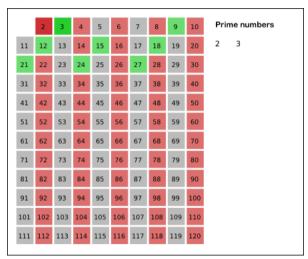


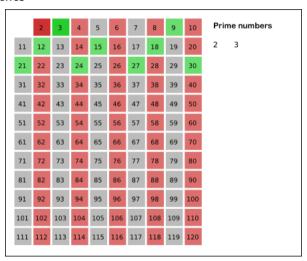


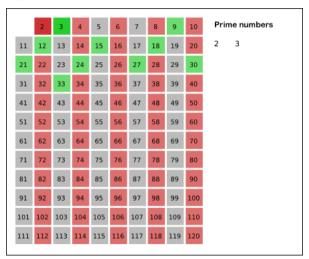


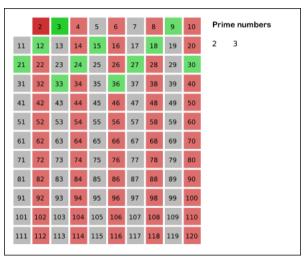


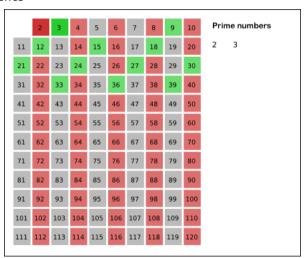


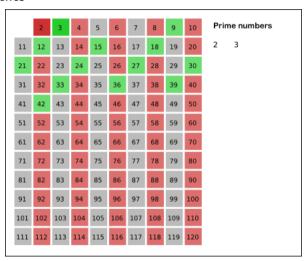


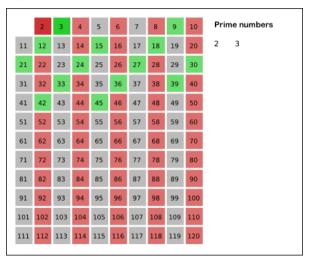


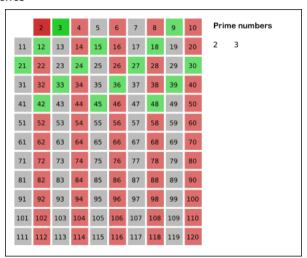


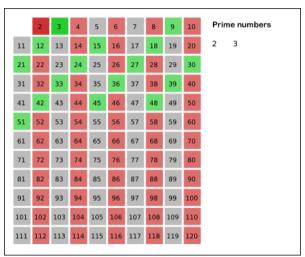


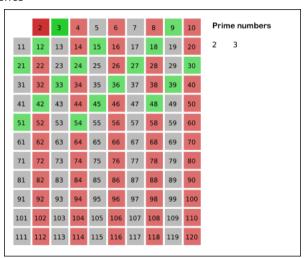




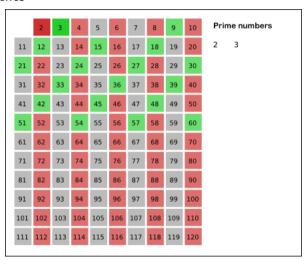


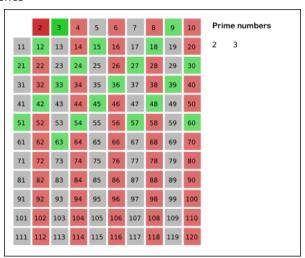


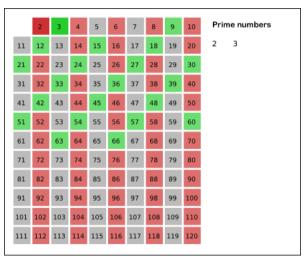


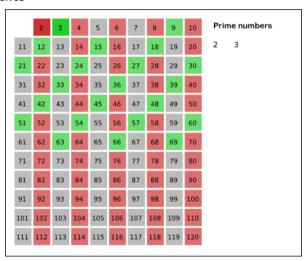


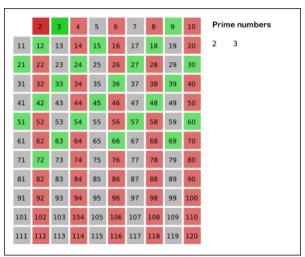


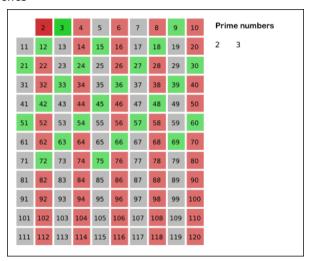


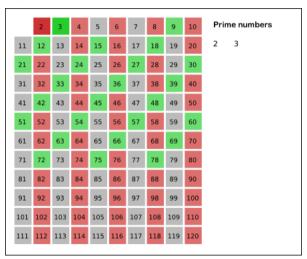


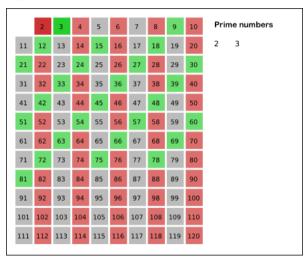


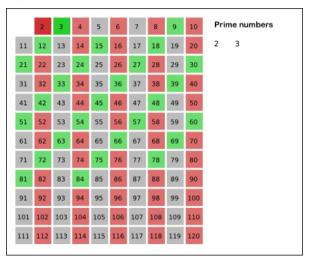


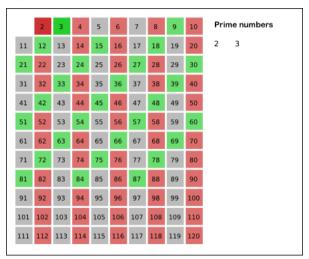


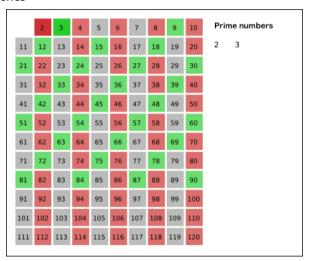


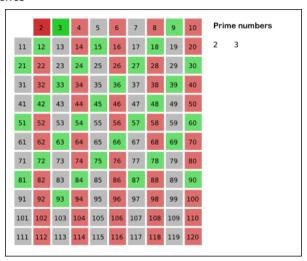




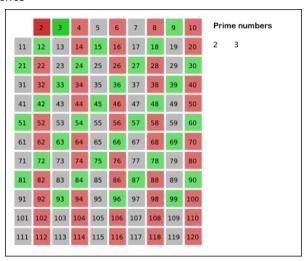


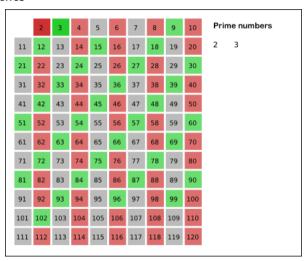


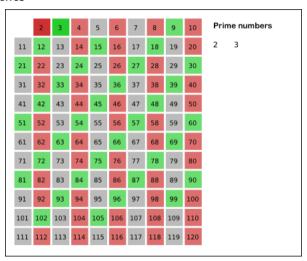


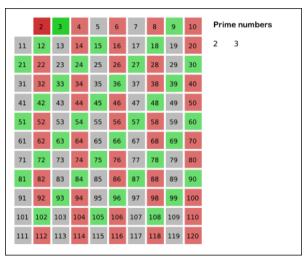


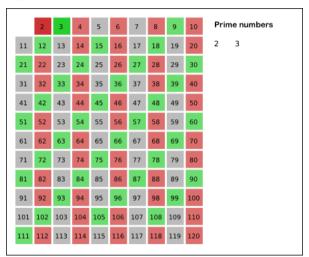


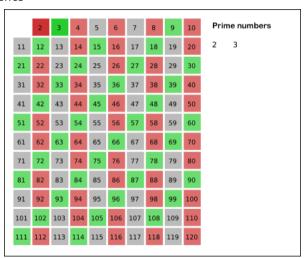


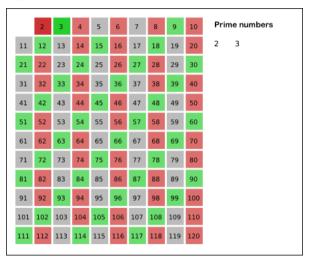


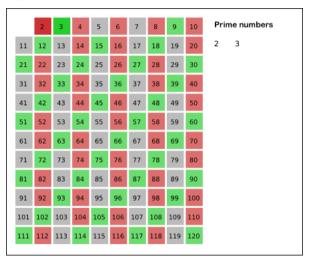


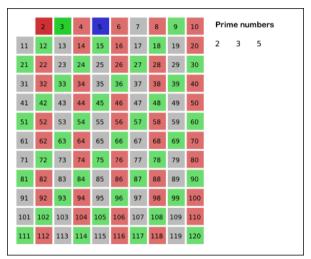


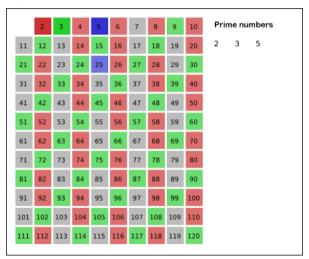


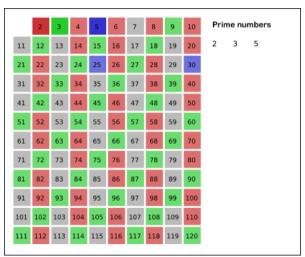


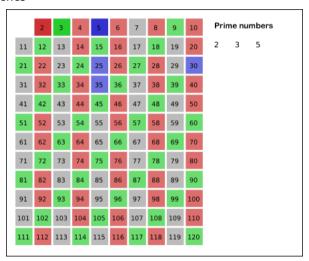


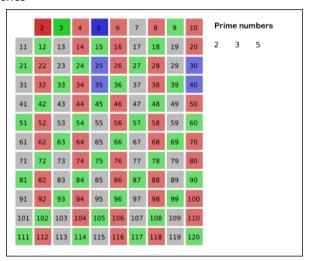


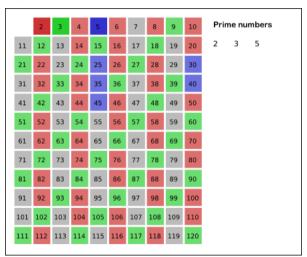


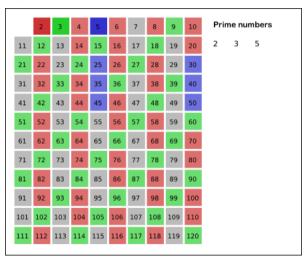


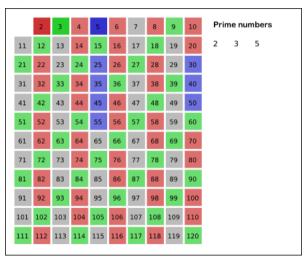


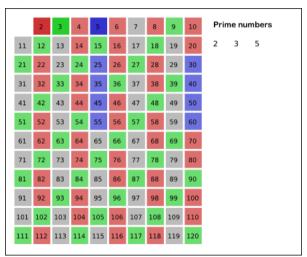


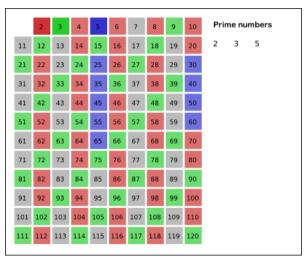


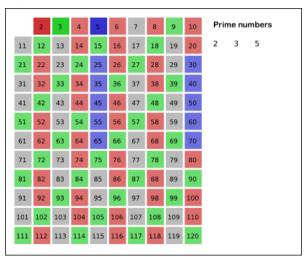


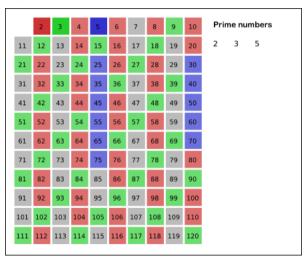


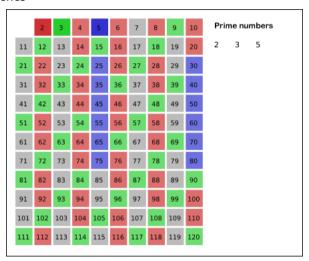


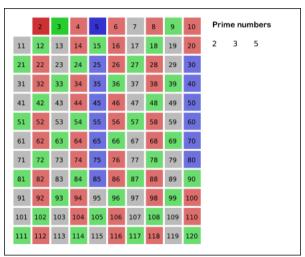


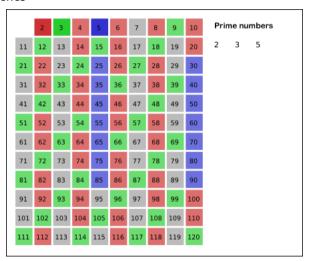


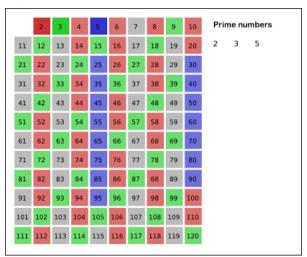


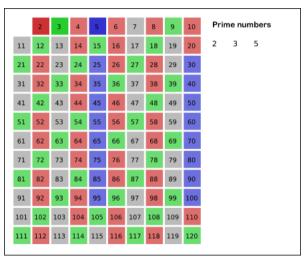


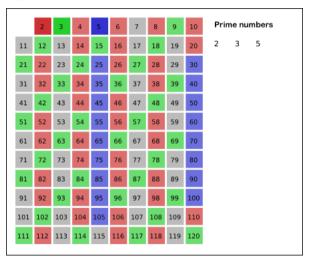


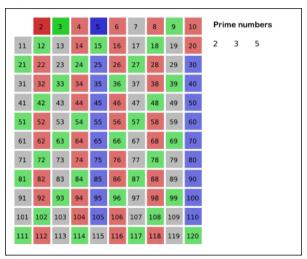


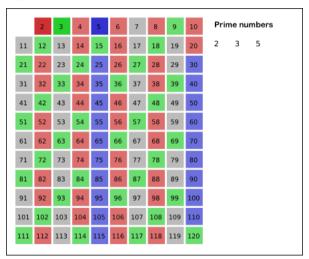


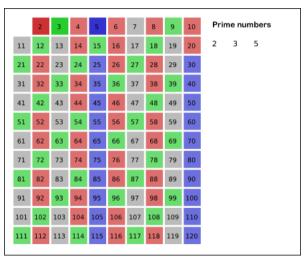


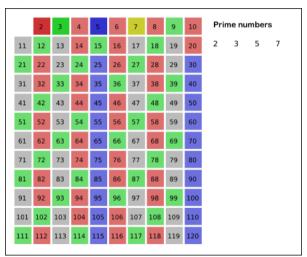


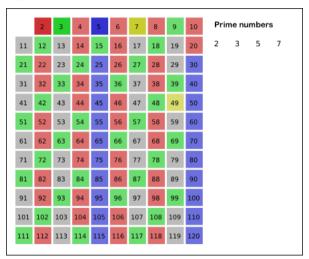


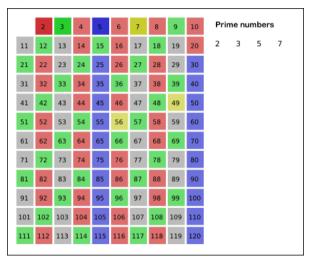


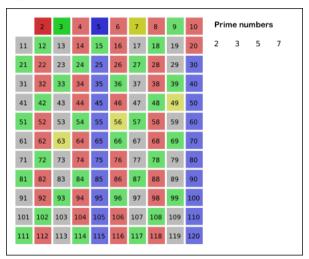


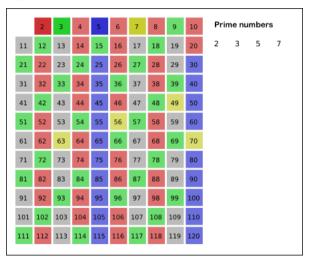


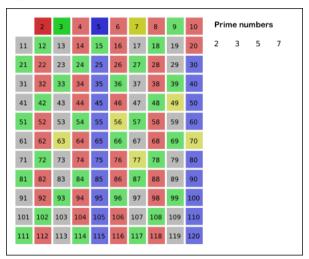


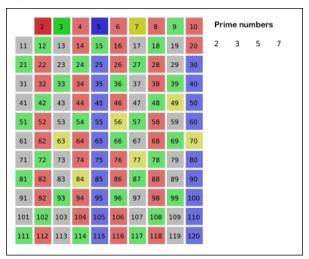


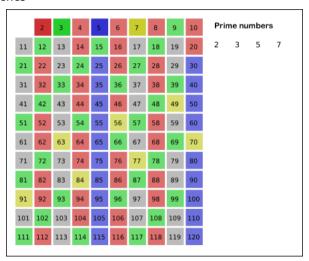


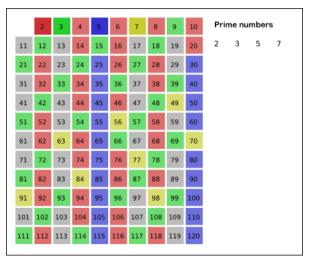


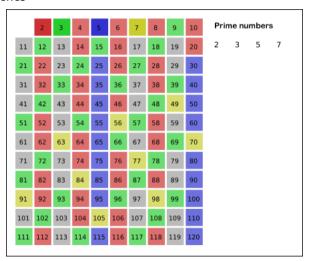


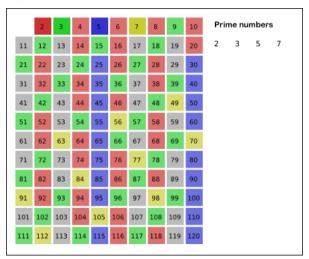


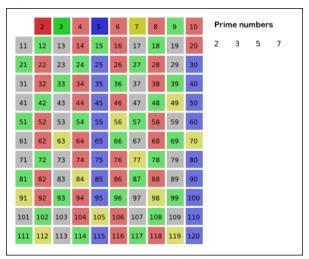


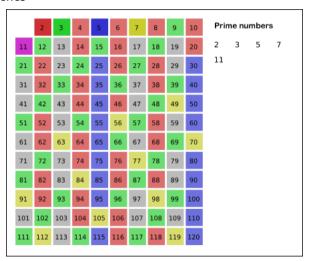


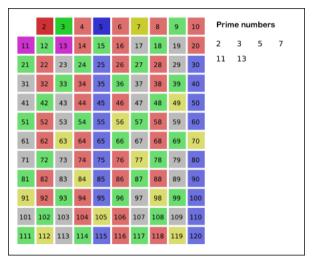


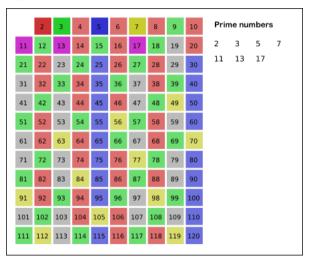


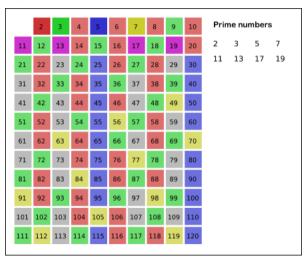


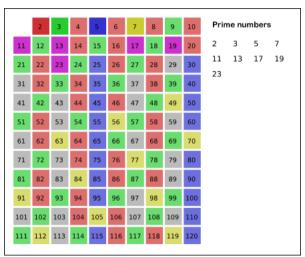


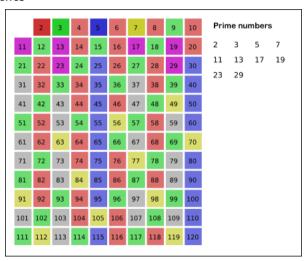


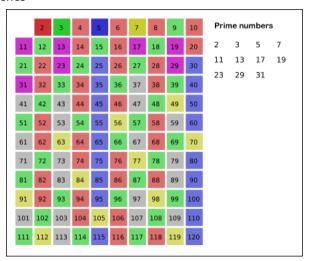


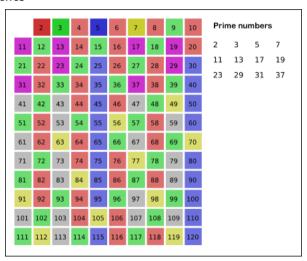


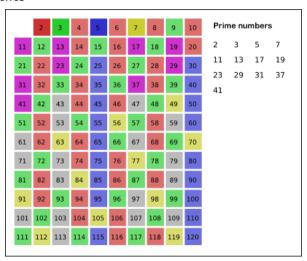


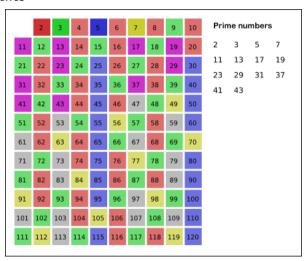


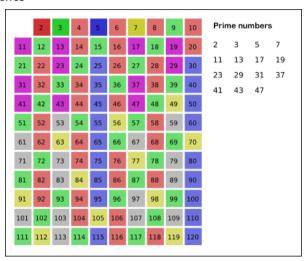


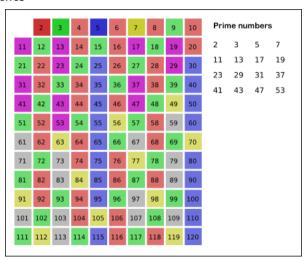


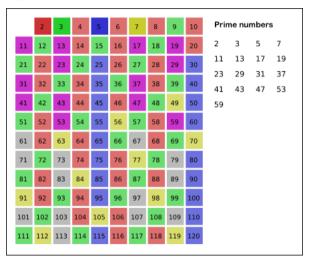


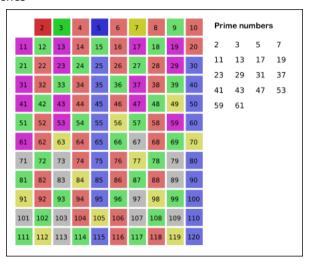


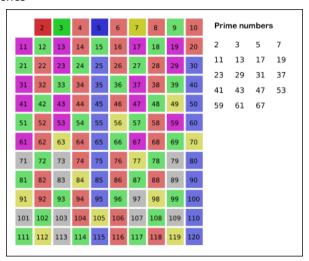


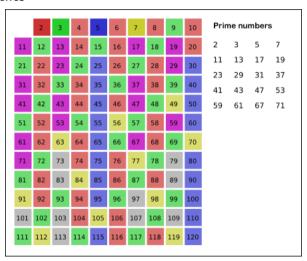


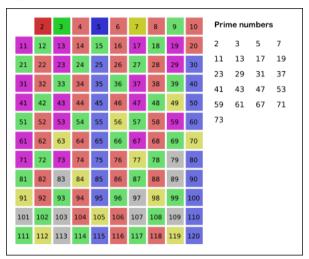


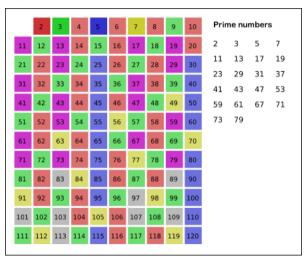


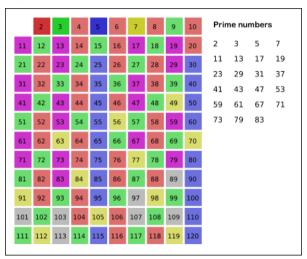


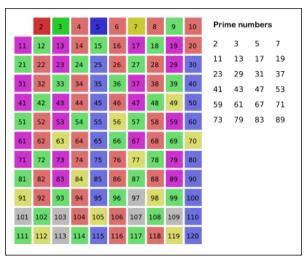


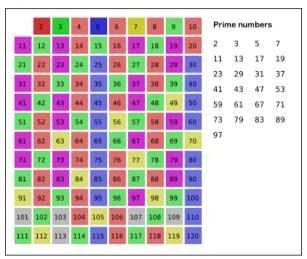


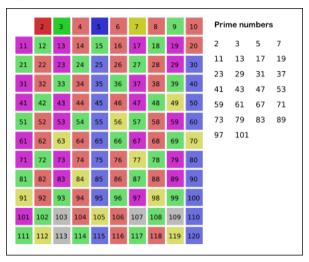


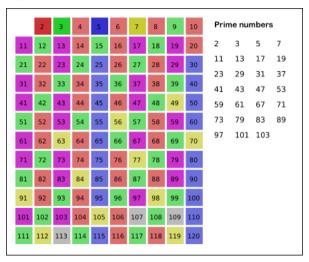


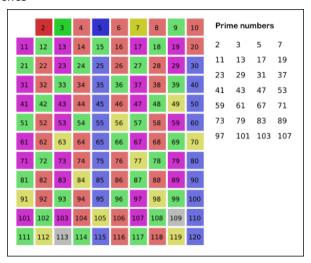


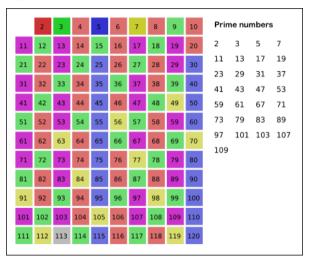


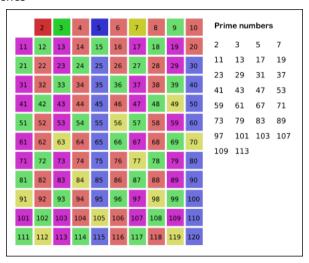












```
import numpy as np
import numba as nb
@nb.njit # simply add the njit decorator
def primes(max=100000):
   numbers = np.ones(max, dtype=np.uint8) # initialize the boolean sieve
   for i in range(2, max):
        if numbers[i] == 0: # has previously been crossed off
           continue
        else: # it is a prime, cross off all multiples
           x = i + i
           while x < max:
               numbers[x] = 0
               v += i
    # return all primes, as indicated by all boolean positions that are one
   return np.nonzero(numbers)[0][2:]
```

```
In [5]: "timeit sieve.primes.py_func()

124 ms ± 2.72 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [6]: "timeit sieve.primes()

308 µs ± 8.93 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

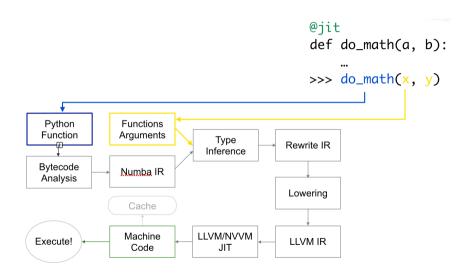
### **Open Source Status**

- Several companies and many open source contributors
  - 4 FTE funded to contribute to Numba between Anaconda, Intel, Quansight, and others
  - 7-12 non-core contributors per release ( every 2 months)
- Slowly moving towards NumFocus application
- Issue and PR lists growing

### Outline

- Introduction
- 2 Going Deeper
- NumPy Support
- 4 Tips and Tricks
- **5** Summary?

### Numba Flow



#### Internals

- Translate Python objects of supported types into representations with no CPython dependencies ("unboxing")
- Compile Python bytecode from your decorated function into machine code.
- Swap calls to builtins and NumPy functions for implementations provided by Numba (or 3rd party Numba extensions)
- Allow LLVM to inline functions, autovectorize loops, and do other optimizations you would expect from a C compiler
- Allow LLVM to exploit all supported instruction sets of your hardware (SSE, AVX)
- When calling the function, release the GIL if requested
- Convert return values back to Python objects ("boxing")

### What Numba does not do

- Automated translation of CPython or NumPy implementations
- Automatic compilation of 3rd party libraries
- Partial compilation
- Automatic conversion of arbitrary Python types
- Change the layout of data allocated in the interpreter
- Translate entire programs
- Magically make individual NumPy functions faster

# When is Numba unlikely to help?

- Whole program compilation
- Critical functions have already been converted to C or optimized Cython
- Need to interface directly to C++
- Algorithms are not primarily numerical
- Exception: Numba can do pretty well at bit manipulation

### Outline

- Introduction
- 2 Going Deeper
- NumPy Support
- 4 Tips and Tricks
- **5** Summary?

# Implementing Numpy

- Numba does not use any of the NumPy C implementations
- We implement the NumPy API using Numba compatible/supported Python
- Treat Numpy as DSL for array oriented computing

# Implementing Numpy

- Implement: numpy.linalg.norm
- For vectors, ord is:
  - $\inf \rightarrow \min(abs(x))$
  - 0  $\rightarrow$  sum(x != 0)
- Implemented in: numba.targets.linalg.py

# Implementing Numpy

```
elif ord == -np.inf:
    # min(abs(a))
    ret = abs(a[0])
    for k in range(1, n):
        val = abs(a[k])
        if val < ret:</pre>
           ret = val
    return ret
elif ord == 0:
    \# sum(a != 0)
    ret = 0.0
    for k in range(n):
        if a[k] != 0.:
           ret += 1.
    return ret
```

### Outline

- Introduction
- 2 Going Deeper
- NumPy Support
- Tips and Tricks
- **5** Summary?

# Tips and Tricks

- Always use jit(nopython=True)
- (Or njit)
- Prefer Numpy arrays for numerical data
- Use typed containers from numba.typed for nested data
- Use small functions, they are inlined
- for loops are fine
- Array expressions are fused, so those are fine too

# Typed Containers

```
from numba import njit
from numba.typed import List
1 = List() # instantiate a new typed list
11 = List(); [11.append(i) for i in range(5)]
1.append(11) # add the first sub-list
12 = List(); [12.append(i) for i in range(10)]
1.append(12) # add the second sub-list
print(1)
@njit
def func(my_list):
    # modify list in a compiled context
    for i in range(10):
        my_list[1][i] = 23
func(1)
print(1)
```

# **Typed Containers**

```
$ python code/typed.py
[[0, 1, 2, 3, 4], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]]
[[0, 1, 2, 3, 4], [23, 23, 23, 23, 23, 23, 23, 23, 23]]
```

# **Fused Expressions**

```
import numpy as np
from numba import njit
a, b = np.arange(1e6), np.arange(1e6)
@njit
def func(a, b):
    return a*b-4.1*a > 2.5*b
```

### **Fused Expressions**

```
In [1]: from fused import a,b,func

In [2]: %timeit func.py_func(a,b)
4.68 ms ± 89.7 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

In [3]: %timeit func(a,b)
626 µs ± 22.4 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

# OS Support

- Windows 7 and later, 32 and 64-bit
- macOS 10.9 and later, 64-bit
- Linux (most anything greater than RHEL 5), 32-bit and 64-bit

# Python versions

- Python 2.7, 3.4-3.7
- NumPy 1.10 and later

# Hardware Support

- x86, x86\_64|AMD64 CPUs
- NVIDIA CUDA GPUs (Compute capability 3.0 and later, CUDA 8.0 and later)
- AMD GPUs (ROCm on Linux)
- ARM 32-bit (Raspbery Pi) and 64-bit (Jetson TX2)
- POWER8|9

# Packaging

- You can depend on Numba to perform the heavy lifting!
- We run CI on most OS/Python/Hardware combinations
- You can ship a single source package
  - PyPi
  - anaconda.org
- No need to pre-compile binaries for your users

### Outline

- Introduction
- 2 Going Deeper
- NumPy Support
- 4 Tips and Tricks
- Summary?

### **Understood Numba?**

- Numba is a
  - just-in-time
  - type-specializing
  - function compiler
  - for accelerating numerically-focused Python

# What is Cooking?

- Dropping Python 2.7 support
- ullet Python o Assembly visualizer
- Automatic inspection of Numpy Support
- Scipy + Numba (aka Scumba)
- Numba-integration-testing (TexasBBQ)
- AirDask-Cl

# Getting in Touch

- https://numba.pydata.org
- Preferred: GitHub + Gitter
- Stuck? It can't hurt to ask. ;-)