Sesión 5
Pandas

Andrés Masegosa
Curso: Introducción a Python
Almería, 11 Abril 2019

Python no es un lenguaje para computación científica!

… Python es un *pegamento*.

Python une un conjunto de herramientas científicas.

Una sintaxis de alto nivel que *envuelve* bibliotecas escritas en C/Fortran de bajo nivel, que es donde ocurre el cálculo.

Es la **velocidad de desarrollo**, no necesariamente la **velocidad de ejecución**, lo que ha impulsado la popularidad de Python.

CDTIME

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

Assembler

```
1   global _mult3
2   sum equ 16
3   section .text
4   _mult3:
5     push ebp
6     mov ebp,esp
7     push esi
8     push edi
9     sub esp, 4
10    mov esi, [ebp+12]
11    mov edi, [ebp+8]
12    mov dword [ebp-sum], 0
13    mov ecx, 3
14  .forloop:
15    mov eax, [edi]
16    imul dword [esi]
17    add edi, 4
18    add esi, 4
19    add [ebp-sum], eax
20    loop .forloop
21    mov eax, [ebp-sum]
22    add esp, 4
23    pop edi
24    pop esi
25    pop ebp
26    ret
```

C source[1]

```
1   int mult3( int *dst, int *src)
2   {
3     int sum = 0, i;
4
5     for (i = 0; i < 3; i++)
6       sum += dst[i] * src[i];
7
8     return sum;
9   }
10
11  int main(void)
12  {
13    int d[3] = { 1, 2, 3};
14    int s[3] = {8, 9, 10 };
15
16    printf("answer is %i\n", mult3(d, s) );
17    return 0;
18  }
```

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

Python

```
1  import numpy
2  dist = numpy.array([1,2,3])
3  src  = numpy.array([8,9,10])
4
5  print dist.dot(src)
```

CDTIME

SCIENTIFIC PYTHON

Python: An Ecosystem for Scientific Computing

As the relationship between research and computing evolves, new tools are required to not only treat numerical problems, but also to solve various problems that involve large datasets in different formats, new algorithms, and computational systems such as databases and Internet servers. Python can help develop these computational research tools by providing a balance of clarity and flexibility without sacrificing performance.

Scientific computing, a discipline at the intersection of scientific research, engineering, and computing, has traditionally focused either on raw performance (in languages such as Fortran and C/C++) or generality and ease of use (in systems such as Matlab or Mathematica), and mainly for numerical problems. Today, scientific researchers use computers for problems that extend far beyond pure numerics, and we need tools flexible enough to address issues beyond performance and usability.

As we describe here, the Python programming language, augmented with a stack of open source tools developed over the past decade by a diverse group of scientists and engineers, provides a *computational ecosystem* that is quite capable of tackling these new challenges.

Scientific Computing's Changing Landscape

For a long time, scientific computing was focused almost exclusively on raw performance for floating-point numerical tasks. It's no accident that Fortran is an abbreviation of *formula translation*: for a long time, computing numerical formulas was a computer's main purpose. Today, however, scientific computing's algorithmic needs go far beyond floating-point numerics. Despite the lasting importance and usefulness of array-oriented libraries such as Lapack (and its many descendants), modern scientific codes routinely tackle problems that go beyond number crunching with arrays.

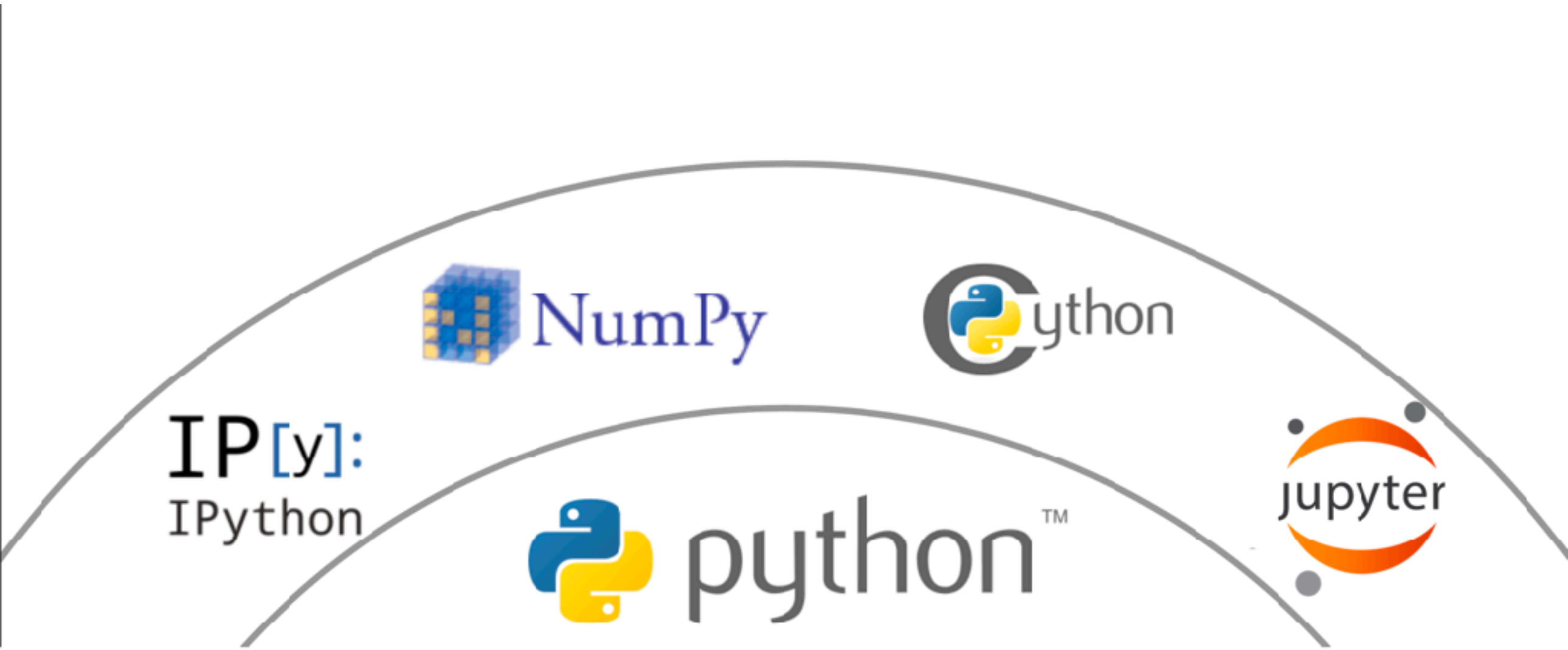
Problems such as graph traversal, finite state machines, or branch and bound algorithms are now a staple of many scientific codes.[1] These require data structures beyond simple arrays and use code full of logic and integer manipulations that is quite different from what tools like Lapack are good at. In addition, even today's numerical work has needs beyond hardware floating point, as many current problems of interest require integrated access to arbitrary precision integers, rationals, and floating-point numbers, often in combination with symbolic manipulation.

The Python tool stack doesn't attempt to replace the many critical codes and algorithms with versions written in Python. Rather, the approach is to expose those codes through Python wrappers while providing rich data structures and programming paradigms to tackle problems not easily managed with high-performance computing's traditional data structures. Although Python isn't unique in possessing rich data structures (C++ has them as well), it's particularly good at

Fernando Perez
University of California, Berkeley
Brian E. Granger
California Polytechnic State University, San Luis Obispo
John D. Hunter
TradeLink Securities

http://www.ericmajinglong.com/wp-content/uploads/2015/01/A19C7DF8-31E5-4984-8045-D17A3C963C55.pdf


python™


NumPy
Cython
IP[y]:
IPython
python™
jupyter

matplotlib

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

SciPy

NumPy

Cython

SymPy

IP[y]:
IPython

python™

jupyter

CDTIME

PyTables

NetworkX

SM StatsModels
Statistics in Python

scikits-image
image processing in python

scikit learn

matplotlib

pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

PyMC

SciPy

NumPy

Cython

SymPy

IP[y]:
IPython

python™

jupyter

CDTIME

PYRO

InferPy

TensorFlow

mxnet

PyTorch

Microsoft Cognitive Toolkit

NumPy

Cython

IP[y]: IPython

python™

jupyter

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$