Python for Scientific Computing

Eric Jones
Language Popularity (as rated by TIOBE*)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>↑</td>
<td>Java</td>
<td>18.160%</td>
<td>+0.20%</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>↓</td>
<td>C</td>
<td>16.170%</td>
<td>-2.02%</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>=</td>
<td>C++</td>
<td>9.146%</td>
<td>-1.23%</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>↑↑↑↑</td>
<td>C#</td>
<td>7.539%</td>
<td>+2.76%</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>↓</td>
<td>PHP</td>
<td>6.508%</td>
<td>-2.57%</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>↑↑↑↑</td>
<td>Objective-C</td>
<td>5.010%</td>
<td>+2.65%</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>=</td>
<td>Python</td>
<td>4.583%</td>
<td>+0.49%</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>↓↓↓</td>
<td>(Visual) Basic</td>
<td>4.496%</td>
<td>-1.16%</td>
<td>A</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>↓</td>
<td>Perl</td>
<td>2.231%</td>
<td>-1.05%</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>↑</td>
<td>Ruby</td>
<td>1.421%</td>
<td>-0.67%</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>↑</td>
<td>JavaScript</td>
<td>1.394%</td>
<td>-0.69%</td>
<td>A</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
<td>↑↑↑↑↑</td>
<td>Lua</td>
<td>1.102%</td>
<td>+0.61%</td>
<td>A</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>↓↓↓</td>
<td>Delphi</td>
<td>1.073%</td>
<td>-1.49%</td>
<td>A</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>=</td>
<td>Assembly</td>
<td>1.042%</td>
<td>-</td>
<td>A</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>↑</td>
<td>Lisp</td>
<td>0.953%</td>
<td>+0.30%</td>
<td>A</td>
</tr>
<tr>
<td>16</td>
<td>23</td>
<td>↑↑↑↑↑</td>
<td>Ada</td>
<td>0.747%</td>
<td>+0.32%</td>
<td>A</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>↓↓↓</td>
<td>Pascal</td>
<td>0.709%</td>
<td>-0.02%</td>
<td>A</td>
</tr>
<tr>
<td>18</td>
<td>21</td>
<td>↑↑↑</td>
<td>Transact-SQL</td>
<td>0.697%</td>
<td>+0.21%</td>
<td>B</td>
</tr>
<tr>
<td>19</td>
<td>-</td>
<td>=</td>
<td>Scheme</td>
<td>0.580%</td>
<td>-</td>
<td>B</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td>↑↑↑↑</td>
<td>RPG (OS/400)</td>
<td>0.503%</td>
<td>+0.09%</td>
<td>B</td>
</tr>
</tbody>
</table>

*www.tiobe.com*
# Language Popularity (as rated by TIOBE*)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>↑</td>
<td>Java</td>
<td>18.160%</td>
<td>+0.20%</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>↓</td>
<td>C</td>
<td>16.170%</td>
<td>-2.02%</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>=</td>
<td>C++</td>
<td>9.146%</td>
<td>-1.23%</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>↑↑↑</td>
<td>C#</td>
<td>7.539%</td>
<td>+2.76%</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>↓</td>
<td>PHP</td>
<td>6.508%</td>
<td>-2.57%</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>↑↑↑↑↑↑↑↑</td>
<td>Objective-C</td>
<td>5.010%</td>
<td>+2.65%</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>=</td>
<td>Python</td>
<td>4.583%</td>
<td>+0.49%</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>↓↓↓</td>
<td>(Visual) Basic</td>
<td>4.496%</td>
<td>-1.16%</td>
<td>A</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>↓</td>
<td>Perl</td>
<td>2.231%</td>
<td>-1.05%</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>↑</td>
<td>Ruby</td>
<td>1.421%</td>
<td>-0.67%</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>↑</td>
<td>JavaScript</td>
<td>1.394%</td>
<td>-0.69%</td>
<td>A</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
<td>↑↑↑↑↑↑↑↑↑</td>
<td>Lua</td>
<td>1.102%</td>
<td>+0.61%</td>
<td>A</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>↓↓↓</td>
<td>Delphi</td>
<td>1.073%</td>
<td>-1.49%</td>
<td>A</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>=</td>
<td>Assembly</td>
<td>1.042%</td>
<td>-</td>
<td>A</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>↑</td>
<td>Lisp</td>
<td>0.953%</td>
<td>+0.30%</td>
<td>A</td>
</tr>
<tr>
<td>16</td>
<td>23</td>
<td>↑↑↑↑↑↑↑↑↑</td>
<td>Ada</td>
<td>0.747%</td>
<td>+0.32%</td>
<td>A</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>↓↓↓</td>
<td>Pascal</td>
<td>0.709%</td>
<td>-0.02%</td>
<td>A</td>
</tr>
<tr>
<td>18</td>
<td>21</td>
<td>↑↑↑↑↑</td>
<td>Transact-SQL</td>
<td>0.697%</td>
<td>+0.21%</td>
<td>B</td>
</tr>
<tr>
<td>19</td>
<td>-</td>
<td>=</td>
<td>Scheme</td>
<td>0.580%</td>
<td>-</td>
<td>B</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td>↑↑↑↑↑↑↑</td>
<td>RPG (C/400)</td>
<td>0.503%</td>
<td>+0.09%</td>
<td>B</td>
</tr>
</tbody>
</table>

* www.tiobe.com
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>↑</td>
<td>Java</td>
<td>18.160%</td>
<td>+0.20%</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>↓</td>
<td>C</td>
<td>16.170%</td>
<td>-2.02%</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>=</td>
<td>C++</td>
<td>9.146%</td>
<td>-1.23%</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>↑↑↑↑↑↑</td>
<td>C#</td>
<td>7.539%</td>
<td>+2.76%</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>↓</td>
<td>PHP</td>
<td>6.508%</td>
<td>-2.57%</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>↑↑↑↑↑↑↑↑↑</td>
<td>Objective-C</td>
<td>5.010%</td>
<td>+2.65%</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>=</td>
<td>Python</td>
<td>4.583%</td>
<td>+0.49%</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>↓↓↓↓↓↓↓↓↓</td>
<td>(Visual) Basic</td>
<td>4.496%</td>
<td>-1.16%</td>
<td>A</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>↓</td>
<td>Perl</td>
<td>2.231%</td>
<td>-1.05%</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>↑</td>
<td>Ruby</td>
<td>1.421%</td>
<td>-0.67%</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>↑</td>
<td>JavaScript</td>
<td>1.394%</td>
<td>-0.69%</td>
<td>A</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
<td>↑↑↑↑↑↑↑↑↑</td>
<td>Lua</td>
<td>1.102%</td>
<td>+0.61%</td>
<td>A</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>↓↓↓↓↓↓↓↓↓</td>
<td>Delphi</td>
<td>1.073%</td>
<td>-1.49%</td>
<td>A</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>=</td>
<td>Assembly</td>
<td>1.042%</td>
<td>-</td>
<td>A</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>↑</td>
<td>Lisp</td>
<td>0.953%</td>
<td>+0.30%</td>
<td>A</td>
</tr>
<tr>
<td>16</td>
<td>23</td>
<td>↑↑↑↑↑↑↑↑↑</td>
<td>Ada</td>
<td>0.747%</td>
<td>+0.32%</td>
<td>A</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>↓↓↓↓↓↓↓↓↓</td>
<td>Pascal</td>
<td>0.709%</td>
<td>-0.02%</td>
<td>A</td>
</tr>
<tr>
<td>18</td>
<td>21</td>
<td>↑↑↑↑↑↑↑↑↑</td>
<td>Transact-SQL</td>
<td>0.697%</td>
<td>+0.21%</td>
<td>B</td>
</tr>
<tr>
<td>19</td>
<td>-</td>
<td>=</td>
<td>Scheme</td>
<td>0.580%</td>
<td>-</td>
<td>B</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td>↑↑↑↑↑↑↑↑↑</td>
<td>RPG (COBOL)</td>
<td>0.503%</td>
<td>+0.09%</td>
<td>B</td>
</tr>
</tbody>
</table>

*www.tiobe.com*
Scripting Layer in Applications
Scripting Layer in Applications

```python
>>> # Create data objects
test1 = TestDevice()

>>> # Sweep voltage and look at Bit Error Rate
>>> voltages = linspace(1, 2, 11)
>>> error_rates = []
>>> for voltage in voltages:
...   error_rate = test1.error_rate()
...   error_rates.append(error_rate)

>>> error_rates = array(error_rates)
>>> # Plot the results
>>> plot(voltages, error_rates)
```
VMS – Virtual Mixing System

- Design Drawings
- Computational Fluid Dynamics
- Parallel Simulation
- Data Visualization
Repository designer
AeroCity - Next Generation Wing Design
Dental Surgery Planning
Python Ecosystem

3D data vis

numerics

GUI

500 MB data file

parallel execution

ORM

2D data vis

linear algebra

statistical analysis
"Per Trace" Windowed Algorithm
# Incorporating Algorithms: NumPy

```python
Python and NumPy

# Calculate "some windowed algorithm" along an axis for an input data cube.

def some_algorithm(input, output, window=7):
    fs = int((window - 1) / 2.0)

    rng0 = range(fs, input.shape[0] - fs + 1)

    tmp = zeros(output.shape[1:])
    for rk in rng0:
        output[rk, :, :] = 0
        tmp[:, :] = 0
        rng = range(rk - fs, rk + fs , 2)
        for wk in rng:
            <snip>
```
Incorporating Algorithms: C/C++

# Calculate "some windowed algorithm" along an axis for an input data cube.

```python
def some_algorithm(input, output, window=7):
    main_code = ""
    int Nx=Ninput[0]; int Ny=Ninput[1]; int Nz=Ninput[2];

    int wlow=-window / 2;
    int whigh=window + wlow;
    int xx, tmp;

    // Using symmetric window and reflecting at the boundaries
    for(int x=0; x < Nx; ++x) {
        for(int y=0; y < Ny; ++y) {
            for(int z=0; z < Nz; ++z) {
                tmp = 0;
                for(int w=wlow; w<=whigh; w+=2) {
                    <snip>
                }
                tmp /= window;
            }
        }
    }
    ""
    weave.inline(main_code, ['input', 'output', 'window'])
```

Weave and C/C++
Incorporating Algorithms: Fortran

```fortran
subroutine algorithm (input, output, window, nx, ny, nz)
    integer window, nx, ny, nz
    integer input(nx, ny, nz)
    integer output(nx, ny, nz)

    integer wlow, whigh, tmp, x, y, z, xx, w
    wlow = -window / 2
    whigh = window + wlow

    do z=1, nz
        do y=1, ny
            do x=1, nx
                tmp=0
                do w=wlow, whigh, 2
                    tmp = tmp + input(x+w, y+w, z+w)
                enddo
                output(x, y, z) = tmp
            enddo
        enddo
    enddo
    return
end
```

```bash
>>> f2py -c -m _algorithm_fortran algorithm_fortran.f
```

f2py and Fortran
# Calculate "some windowed algorithm" along an axis for an input data cube.

def algorithm(input, output, window=7):
    algorithm_kernel = '''
    __kernel void algorithm(__global const int* input, __global int* output, const int Nx, const int Ny, const int Nz, const int window)
    {
        int x=get_global_id(0), y=get_global_id(1), z=get_global_id(2);
        int xx, tmp = 0;
        for(int w=-window/2; w<=window-window/2; w+=2) {
            <snip>
        }
        output[(x*Ny+y)*Nz+z] = tmp / window;
    }'''

    ctx = cl.create_some_context(0)
    queue = cl.CommandQueue(ctx)
    mf = cl.mem_flags
    # Create buffers on device
    input_buf = cl.Buffer(ctx, mf.READ_ONLY|mf.COPY_HOST_PTR, hostbuf=input)
    output_buf = cl.Buffer(ctx, mf.WRITE_ONLY, output.nbytes)
    # "compile" the program
    prg = cl.Program(ctx, activity_kernel).build()
    # run the program
    Nx, Ny, Nz = map(np.int32, input.shape)
    prg.algorithm(queue, output.shape, None, input_buf, output_buf, Nx, Ny, Nz, np.int32(window))
    # This incurs some overhead
    cl.enqueue_copy(queue, output, output_buf).wait()
## Flexibility and Speed

<table>
<thead>
<tr>
<th>Technology</th>
<th>Time (sec)</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python+NumPy</td>
<td>0.66</td>
<td>1.00</td>
</tr>
<tr>
<td>C++ &amp; Weave</td>
<td>0.35</td>
<td>1.89</td>
</tr>
<tr>
<td>Fortran &amp; f2py</td>
<td>0.49</td>
<td>1.35</td>
</tr>
<tr>
<td>GPU: PyOpenCL</td>
<td>0.42</td>
<td>1.57</td>
</tr>
<tr>
<td>PyOpenCL (no copy)</td>
<td>0.026</td>
<td>25.4</td>
</tr>
</tbody>
</table>

Algorithm and Data: window size=7, input=200x200x500 int32 cube.*

Hardware: Macbook Pro 2.66 GHz, 8 GB RAM, NVidia GeForce 330M

* Results vary WIDELY with array size.
Memory Mapped Example

• # Create a "memory mapped" array where
• # the array data is stored in a file on
• # disk instead of in main memory.
• >>> from numpy import memmap
• >>> image = memmap('some_file.dat',
       dtype=uint16,
       mode='r+',
       shape=(5,5),
       offset=header_size)

• # Standard array methods work.
• >>> mean_value = image.mean()

• # Standard math operations work.
• # The resulting scaled_image *is*
• # stored in main memory. It is a
• # standard numpy array.
• >>> scaled_image = image * .5
### Memmap Timings (3D arrays)

<table>
<thead>
<tr>
<th>Operations (500x500x1000)</th>
<th>Linux</th>
<th>OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In Memory</td>
<td>Memory Mapped</td>
</tr>
<tr>
<td>read</td>
<td>2103 ms</td>
<td>11.0 ms</td>
</tr>
<tr>
<td>x slice</td>
<td>1.8 ms</td>
<td>4.8 ms</td>
</tr>
<tr>
<td>y slice</td>
<td>2.8 ms</td>
<td>4.6 ms</td>
</tr>
<tr>
<td>z slice</td>
<td>9.2 ms</td>
<td>13.8 ms</td>
</tr>
<tr>
<td>downsample 4x4</td>
<td>0.02 ms</td>
<td>125 ms</td>
</tr>
</tbody>
</table>

All times in milliseconds (ms).

Linux: Ubuntu 4.10, Dell Precision 690, Dual Quad Core Zeon X5355 2.6 GHz, 8 GB Memory
OS X: OS X 10.5, MacBook Pro Laptop, 2.6 GHz Core Duo, 4 GB Memory
IPython – Embarrassingly Parallel

Master
Python >>>

Socket

Node 0
Python >>>

Node 1
Python >>>

Node 2
Python >>>
Electromagnetic Scattering

Inputs
- environment, target mesh, and multiple frequencies

Mem: KB to Mbytes

Computation
- $N^3$ CPU
- $N^2$ storage
- Time: a few seconds to days

Mem: MB to GBytes

Outputs
- Radar Cross Section values

Mem: KB to MBytes

Monostatic Backscatter from Buried Landmine, Theta = 30, Phi = 0

RCS (dB)

Frequency (MHz)
Parallel FFT On Memory Mapped File

500 MB Data File

<table>
<thead>
<tr>
<th>Processors</th>
<th>Time (seconds)</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.75</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>6.06</td>
<td>1.9</td>
</tr>
<tr>
<td>4</td>
<td>3.36</td>
<td>3.5</td>
</tr>
<tr>
<td>8</td>
<td>2.50</td>
<td>4.7</td>
</tr>
</tbody>
</table>

```
from numpy import ceil
from ipython1.kernel import client
from geio import vti

# Execute an fft on a sub-section of a seismic cube.
code = ""

from numpy import fft
from geio import vti

seismic, params = vti.read(file_name,rescale=False)
start, end = id*size, (id+1) * size
local_seismic = vti.unclip(seismic[start:end,:,:])
spectrum = fft.fft(local_seismic,axis=-1)

def equal_size_split(ary, cluster):
    # Return the number of rows each worker should work
    return int(ceil(float(len(ary))/len(cluster)))

# Run parallel code on each of the remote processors
file_name = "500_500_1000.vt"
cluster = client.MultiEngineClient(('127.0.0.1',10105))
seismic, params = vti.read(file_name,rescale=False)
cluster["size"] = equal_size_split(seismic, cluster)
cluster["file_name"] = file_name
cluster.execute(code)
```