Scientist meets web dev: how Python became the language of data

Gaël Varoquaux

Inria
Scientist meets web dev: how Python became the language of data

Gaël Varoquaux

Very diverse community

This talk: a reflection on what we have in common, Python

I am talking about things you don’t understand (my science) and things I don’t understand (web dev)
I actually did a PhD in quantum physics

Hence I think I qualify as a “scientist”
I now do computer science for neuroscience

Try to link neural activity to thoughts and cognition
I now do computer science for neuroscience

Try to link neural activity to thoughts and cognition

We attack it as a machine learning problem

Python software: nilearn
On the way, we created a machine-learning library:

scikit-learn
Huge success.
Cool.
Data science is THE thing.
Data science with Python is hot

Huge success.
Cool.
Data science is THE thing.

Python is the go-to language
How did it happen?

We built scikit-learn, others pandas, etc..., but these were built on solid foundations
Scientists come from Jupiter
And web devs from Saturn?
And sysadmins from Neptune?
We’re different

numbers (in arrays)
arrays (of numbers)
arrays
arrays

strings
databases
object-oriented programming
flow control

A bit of a culture gap
Let’s do something together: sort EuroPython site

205 talks:

How OpenStack makes Python better (and vice-versa)
Introduction to aiohttp
So you think your Python startup is worth $10 million...
SQLAlchemy as the backbone of a Data Science company
Learn Python The Fun Way
Scaling Microservices with Crossbar.io
If you can read this you don’t need glasses

Let’s find some common topics with data science

G Varoquaux
Let’s do something together: sort EuroPython site

205 talks:

How OpenStack makes Python better (and vice-versa)

Introduction to aiohttp

So you think your Python startup is worth $10 million...

SQLAlchemy as the backbone of a Data Science company

Learn Python The Fun Way

Scaling Microservices with Crossbar.io

If you can read this you don’t need glasses

Let’s find some common topics with data science
Let’s do something together: sort EuroPython site

205 talks:

- How OpenStack makes Python better (and vice-versa)
- Introduction to aiohttp
- So you think your Python startup is worth $10 million...
- SQLAlchemy as the backbone of a Data Science company
- Learn Python The Fun Way
- Scaling Microservices with Crossbar.io
- If you can read this you don’t need glasses

Let’s find some common topics with data science

import urllib2, bs4
import sklearn, wordcloud

Anyone who has used Python to search text for substring patterns has at least heard of the regular expression module. Many of us use it extensively for parsers and lexers, training gives a quick introduction. This talk will focus on some distinguishing features and exercises to extend your own apps seamlessly. It will cover your plugins, apphooks, toolbar extensions.

G Varoquaux
Let's do something together: sort EuroPython site

Crawl

- the schedule to get a list of titles and URLs
- talk pages to retrieve abstract and tags

bs4: beautiful soup, matchings on the DOM tree
Let’s do something together: sort EuroPython site

**Crawl**
- the schedule to get a list of titles and URLs
- talk pages to retrieve abstract and tags

**bs4**: beautiful soup, matchings on the DOM tree

**Vectorize**

<table>
<thead>
<tr>
<th>Term</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>20</td>
</tr>
<tr>
<td>can</td>
<td>10</td>
</tr>
<tr>
<td>code</td>
<td>4</td>
</tr>
<tr>
<td>is</td>
<td>14</td>
</tr>
<tr>
<td>module</td>
<td>3</td>
</tr>
<tr>
<td>profiling</td>
<td>2</td>
</tr>
<tr>
<td>performance</td>
<td>1</td>
</tr>
<tr>
<td>Python</td>
<td>9</td>
</tr>
<tr>
<td>the</td>
<td>18</td>
</tr>
</tbody>
</table>

Anyone who has used Python to search text for substring patterns has at least heard of the regular expression module. Many of us use it extensively for parsers and lexers, and the **py.test** tool presents a rapid and simple way to write tests for your Python code. This training gives a quick introduction to some distinguishing features. Let’s talk about your plugins, apphooks, toolbar extensions to extend django CMS or how to integrate your own apps seamlessly.
Let’s do something together: sort EuroPython site

Crawl

- the schedule to get a list of titles and URLs
- talk pages to retrieve abstract and tags

bs4: beautiful soup, matchings on the DOM tree

Vectorize

Anyone who has used Python to search text for substring patterns has at least heard of the regular expression module. Many of us use it extensively for parsers and lexers, and training gives a quick introduction into some distinguishing features.

Chat with the core developers about how to extend django CMS or how to integrate your own apps seamlessly. Let’s talk about your plugins, apphooks, toolbar extensions, profiling, and performance.

<table>
<thead>
<tr>
<th>Term</th>
<th>Freq</th>
<th>All docs</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>20</td>
<td>1321</td>
</tr>
<tr>
<td>can</td>
<td>10</td>
<td>540</td>
</tr>
<tr>
<td>code</td>
<td>4</td>
<td>208</td>
</tr>
<tr>
<td>is</td>
<td>14</td>
<td>964</td>
</tr>
<tr>
<td>module</td>
<td>3</td>
<td>123</td>
</tr>
<tr>
<td>profiling</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>performance</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Python</td>
<td>9</td>
<td>191</td>
</tr>
<tr>
<td>the</td>
<td>18</td>
<td>1450</td>
</tr>
</tbody>
</table>

G Varoquaux

Term Freq All docs
Let's do something together: sort EuroPython site

**Crawl**
- the schedule to get a list of titles and URLs
- talk pages to retrieve abstract and tags

**bs4**: beautiful soup, matchings on the DOM tree

**Vectorize**

Anyone who has used Python to search text for substring patterns has at least heard of the regular expression module. Many of us use it extensively for parsers and lexers, training gives a quick introduction into some distinguishing features. Let's talk about your plugins, apphooks, toolbar extensions to extend django CMS or how to integrate your own apps seamlessly. Yet again, we can talk profiling and performance.

<table>
<thead>
<tr>
<th>Term</th>
<th>Freq</th>
<th>All docs</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>20</td>
<td>1321</td>
<td>.015</td>
</tr>
<tr>
<td>can</td>
<td>10</td>
<td>540</td>
<td>.018</td>
</tr>
<tr>
<td>code</td>
<td>4</td>
<td>208</td>
<td>.019</td>
</tr>
<tr>
<td>is</td>
<td>14</td>
<td>964</td>
<td>.014</td>
</tr>
<tr>
<td>module</td>
<td>3</td>
<td>123</td>
<td>.023</td>
</tr>
<tr>
<td>profiling</td>
<td>2</td>
<td>7</td>
<td>.286</td>
</tr>
<tr>
<td>performance</td>
<td>1</td>
<td>6</td>
<td>.167</td>
</tr>
<tr>
<td>Python</td>
<td>9</td>
<td>191</td>
<td>.047</td>
</tr>
<tr>
<td>the</td>
<td>18</td>
<td>1450</td>
<td>.012</td>
</tr>
</tbody>
</table>

TF-IDF in scikit-learn

```python
sklearn.feature_extraction.text.TfidfVectorizer
```
Let's do something together: sort EuroPython site

documents
the
Python performance profiling module is code can a

documents
the
Python performance profiling module is code can a

Term-document matrix

G Varoquaux
Let's do something together: sort EuroPython site

documents
the Python performance profiling module is code can a
can code is a

Term-document matrix

Can be a sparse matrix
Let’s do something together: sort EuroPython site

A matrix factorization

Often with non-negative constraints

sklearn.decompositions.NMF

G Varoquaux
Let's do something together: sort EuroPython site

EuroPython abstracts

Topic 1
Let's do something together: sort EuroPython site

EuroPyton abstracts

Topic 2
Let’s do something together: sort EuroPython site

EuroPyton abstracts

Topic 3

G Varoquaux
Let’s do something together: sort EuroPython site

EuroPython abstracts
1. Let's do something together: sort EuroPython site

EuroPyton abstracts

Add one of Python's great templating engine... get a usable website

https://gaelvaroquaux.github.io/my_topics/ep16

G Varoquaux
Want to try it?

$ pip install scikit-learn
Want to try it?

$ pip install scikit-learn
...
ImportError: Numerical Python (NumPy) is not installed.
scikit-learn requires NumPy >= 1.6.1
Want to try it?

$ pip install scikit-learn
...
ImportError: Numerical Python (NumPy) is not installed.
scikit-learn requires NumPy >= 1.6.1

C:> pip install numpy
Want to try it?

$ pip install scikit-learn
...
ImportError: Numerical Python (NumPy) is not installed.
scikit-learn requires NumPy >= 1.6.1

C:> pip install numpy
...
error: Unable to find vcvarsall.bat
We’re different

Well

fast linear algebra

ATLAS (Fortran) 70x faster

libfortran.so.3 ??

you’re kidding me
We’re different

Well

fast linear algebra

ATLAS (Fortran) 70x faster

Packaging is a major roadblock for scientific Python

- A lot of compiled code + shared libraries
  \[\Rightarrow\] library + ABI compatibility issues

Progress:
- Manylinux wheels: PEP 513, RT. McGibbon, NJ. Smith rely on a conservative core set of libs
- Openblas: pure-C, fast linear algebra
But working together gives us awesome things

Text mining $\Rightarrow$ intelligent interfaces
The scientist’s view of code

Numerics versus control flow
Numerics versus databases
Numerics versus strings
Numerics versus the world
Why we love numpy

100,000 term frequency vs inverse doc frequency:

```
In [*]: %timeit [t * i for t, i in izip(tf, idf)]
100 loops, best of 3: 6.2 ms per loop
```

The numpy style:

```
In [*]: %timeit tf * idf
1000 loops, best of 3: 74.2 µs per loop
```
100,000 term frequency vs inverse doc frequency:

In [*]: %timeit [t * i for t, i in izip(tf, idf)]
100 loops, best of 3: **6.2 ms** per loop

The numpy style:

In [*]: %timeit tf * idf
1000 loops, best of 3: **74.2 µs** per loop

Array computing can be more readable

\[tf \times idf\]

vs

\[[t \times i for t, i in izip(tf, idf)]\]
2 arrays are nothing but pointers

A numpy array =

- memory address
- data type
- shape
- strides

Represents any regular data in a structured way: how to access elements via pointer arithmetics (computing offsets)
arrays are nothing but pointers

A numpy array =

- memory address
- data type
- shape
- strides

shape 1

stride 1

stride 2

shape 1

shape 2

Represents any regular data in a structured way:

Matches the memory model of numerical libraries

⇒ Enables copyless interactions

Numpy is really a memory model
Array computing is fast

\[ \text{tf}_\text{idf} = \text{tf} \times \text{idf} \]

- No type checking
- Direct sequential memory access
- Vector operations (SIMD)

![Image of graph showing time per element vs. number of elements for lists and numpy](chart.png)

G Varoquaux
Array computing is limited by CPU starvation

$$\text{tf}\_\text{idf} = \text{tf} \times \text{idf}$$

2x slowdown passed a certain size
Array computing is limited by CPU starvation

\[ \text{tf\_idf} = \text{tf} \times \text{idf} \]

Memory is much slower than CPU

2x slowdown passed a certain size

\[ \log_{10\text{ns}}(10^5) \approx \text{size of the CPU cache} \]
Array computing is limited by CPU starvation.

\[
\text{tfidf} = \text{tf} \times \text{idf} - 1
\]

Memory is much slower than CPU.

It gets worse for complex expressions.
Array computing is limited by CPU starvation

\[ tf_{idf} = tf \times idf - 1 \]

What’s going on:
1. \( \text{tmp} \leftarrow tf \times idf \)
2. \( tf_{idf} \leftarrow \text{tmp} - 1 \)

Big temporary: Moving data in & out of cache

Memory is much slower than CPU
Array computing is limited by CPU starvation

\[ \text{tf}_\text{idf} = \text{tf} \times \text{idf} - 1 \]

What's going on:

1. \( \text{tmp} \leftarrow \text{tf} \times \text{idf} \)
2. \( \text{tf}_\text{idf} \leftarrow \text{tmp} - 1 \)

Big temporary: Moving data in & out of cache

In [1]: %timeit \text{tf} \times \text{idf}
10000 loops, best of 3: 74.2 \mu s per loop

In [2]: %timeit \text{tf} \times \text{idf} - 1
1000 loops, best of 3: 418 \mu s per loop
Array computing is limited by CPU starvation

```
tf_idf = tf * idf - 1
```

What's going on:
1. `tmp ← tf * idf`
2. `tf_idf ← tmp - 1`

Big temporary: Moving data in & out of cache

```
In [*]: %timeit tf * idf
10000 loops, best of 3: 74.2 µs per loop

In [*]: %timeit tf * idf - 1
1000 loops, best of 3: 418 µs per loop
```

**In-place operations**: reuse the allocation

```
In [*]: %timeit tmp = tf * idf; tmp -= 1
10000 loops, best of 3: 112 µs per loop
```
Array computing is limited by CPU starvation

```
tf_idf = tf * idf - 1
```

What’s going on:

1. `tmp ← tf * idf`
2. `tf_idf ← tmp - 1`

Big temporary: Moving data in & out of cache

```
tmp = tf * idf
```

```
tmp -= 1
```
Array computing is limited by CPU starvation

A compilation problem:

tf_idf = tf * idf - 1 \leadsto \text{tf} \cdot \text{idf} = \text{tf} \cdot \text{idf} - 1

1. \text{tmp} \leftarrow \text{tf} \cdot \text{idf}
2. \text{tf_idf} \leftarrow \text{tmp} - 1

Big temporary: Moving data in & out of cache

tmp = tf * idf
\text{tmp} -= 1

```
G Varoquaux 22
```
Array computing is limited by CPU starvation.

A compilation problem:
- Removing/reusing temporaries
- Operating on “chunks” that fit in cache

Addressed by numexpr, with string expressions:
```
numexpr.evaluate('tf * idf - 1', locals())
```

What's going on:
1. \( \text{tmp} \leftarrow \text{tf} \times \text{idf} \)
2. \( \text{tf idf} \leftarrow \text{tmp} - 1 \)

Big temporary: Moving data in & out of cache

A compilation problem:
- Removing/reusing temporaries
- Operating on “chunks” that fit in cache

Addressed by numexpr, with string expressions:
```
numexpr.evaluate('tf * idf - 1', locals())
```

Graph showing performance comparison between NumPy, NumPy inplace, and Numexpr.
Array computing is limited by CPU starvation

A compilation problem:
- Removing/reusing temporaries
- Operating on “chunks” that fit in cache

- Addressed by numexpr, with string expressions 😞
- Addressed by numba, with bytecode inspection 😞
- lazyarray

Similar problem to pagination with SQL queries
Array computing is limited by CPU starvation

$$\text{tf}_\text{idf} = \text{tf} \times \text{idf}$$

TWO SMALL:
OVERHEAD

TWO BIG:
OUT OF CACHE

TIME PER ELEMENT

NUMBER OF ELEMENTS

BIG DATA

G Varoquaux
What if there is an if

\[
\text{tf}_{\text{idf}} = \frac{\text{tf}}{\text{idf}}
\]

\[
\text{tf}_{\text{idf}}[\text{idf} == 0] = 0
\]

Suppose the we are looking at ages in a population:

\[
\text{ages}[\text{gender} == \text{'male']].\text{mean}() - \text{ages}[\text{gender} == \text{'female']].\text{mean}()
\]
**2 Numerics versus control flow**

What if there is an if

```python
tf_idf = tf / idf
tf_idf[idf == 0] = 0
```

Suppose the we are looking at ages in a population:

```python
ages[gender == 'male'].mean() - ages[gender == 'female'].mean()
```

This is really starting to be looking like databases

**pandas: something in between arrays and an in-memory database**

**Great for queries, less great for numerics.**
Installation Problems

Beautiful Python Code

Routines in Fortran or C++

ScalaBility

Deployment Problems

Beautiful Python Code

Database in C++, JAVA, ERLANG...

ScalaBility

Numpy is the scientist's equivalent to an ORM
Gives speed with non-Python code
Numpy is the scientist’s equivalent to an ORM

Gives speed with non-Python code
numerics vs databases

**Numerics**  efficient on regularly spaced data
But numpy creates cache misses for big arrays

⇒ Need to remove temporaries and chunk data
**numerics vs databases**

**numerics**  
efficient on regularly spaced data  
But numpy creates cache misses for big arrays  
⇒ Need to remove temporaries and chunk data

**selection and grouping**  
efficient with indexes or trees  
⇒ Need to group queries

Compilation
**numerics vs databases**

**numerics**
- efficient on regularly spaced data
  
  But numpy creates cache misses for big arrays
  
  ⇒ Need to remove temporaries and chunk data

**selection and grouping**
- efficient with indexes or trees
  
  ⇒ Need to group queries

**Compilation is unpythonic**

A computation & query language?  

<table>
<thead>
<tr>
<th>numexpr</th>
</tr>
</thead>
</table>

I hate domain-specific languages (SQL)

Numpy is very expressive
**numerals vs databases**

**numerals** efficient on regularly spaced data
But numpy creates cache misses for big arrays
⇒ Need to remove temporaries and chunk data

**selection and grouping** efficient with indexes or trees
⇒ Need to group queries

**Compilation is unpythonic**

A computation & query language? **numexpr**
I hate domain-specific languages (SQL)
Numpy is very expressive

PonyORM: Compiling Python to optimized SQL
Datascience with SQL: Ibis & Blaze
**Numerics vs Databases**

**Numerics**
- Efficient on regularly spaced data
- But numpy creates cache misses for big arrays
  => Need to remove temporaries and chunk data

**Selection and Grouping**
- Efficient with indexes or trees
  => Need to group queries

Spark: Java-world “big data” rising star
- Combines distributed store
  - Computing model

We (scikit-learn) are faster when data fits in RAM
Operations on chunks

- Machine learning, data mining $\equiv$ numerics

Out-of-core operations not efficient: no data locality

On-line algorithms (streaming)
Operations on chunks

- Machine learning, data mining $\equiv$ numerics

ETL (extract, transform, & load)

Multivariate statistics
Operations on chunks, or algorithms on chunks

- Machine learning, data mining = numerics

ETL (extract, transform, & load)

Out-of-core operations not efficient: no data locality

On-line algorithms (streaming)

eg stochastic gradient descent
As in deep learning

G Varoquaux
Making the data-science magic happens

from sklearn import

G Varoquaux
Making the data-science magic happens

1. Initialize $\phi_{ni}^0 := 1/k$ for all $i$ and $n$
2. Initialize $\gamma_i := \alpha_i + N/k$ for all $i$
3. Repeat
   4. For $n = 1$ to $N$
      5. For $i = 1$ to $M$
         6. $\beta_{ij} \propto \sum_{d=1}^{N_d} \sum_{n=1}^{N_n} \phi_{dni}^* w_{dn}^i$
         7. $\phi_{ni}^{+1} := \phi_{ni}^{+1}$ normalized to sum to 1
8. $\gamma^{+1} := \gamma + \sum_{n=1}^{N} \phi_{ni}^{+1}$

$\lambda_{ij} = \eta + \sum_{d=1}^{M} \sum_{n=1}^{N_n} \phi_{dni}^* w_{dn}^i$

Turning applied maths papers to robust code
High-level, readable, simple syntax reduces cognitive load

Thanks from sklearn import

G Varoquaux
Beyond numerics

Make #PyData great (again)
Data/computation flow is crucial

Data-flow engines are everywhere

- **dask**
  - pure-Python
  - static compiler
  - dynamic scheduler
  - parallel & distributed

- **theano**
  - expression analysis
  - pure-Python

- **tensorflow**
  - C library
  - distributed

G Varoquaux
Data/computation flow is crucial

Data-flow engines are everywhere

Python should shine there:
reflexivity + metaprogramming + async

“Python is the best numerical language out there because it’s not a numerical language.” – Nathaniel Smith

API challenging:
For algorithm design: no framework / inversion of control
Ingredients for future data flows

Distributed computation & Run-time analysis

Reflexivity is central
- Debugging
- Interactive work
- Code analysis
- Persistence
- Parallel computing

Parallel computing
- Pickle
  - Distribute code and data without data model
  - Serialize intermediate results
  - Deep of hash of any data structure
- joblib.hash
  - Simple parallel syntax:
    - `Parallel(n_jobs=2)(delayed(sqrt)(i) for i in range(10))`
  - Fast persistence:
    - `joblib.dump(anything, 'filename.pkl.gz')`
  - Primitive for out of core:
    - `pointer = mem.cache(f).call`
  - Soon backend system (job broker and persistence)
- Gets job management into algorithms (eg in scikit-learn)
Ingredients for future data flows

Distributed computation & Run-time analysis

Reflexivity is central
- Debugging
  - Interactive work
  - Code analysis
  - Persistence
- Parallel computing

Pickle
- Distribute code and data without data model
- Serialize intermediate results
- Deep of hash of any data structure joblib.hash

Very limited (eg no lambda #19272)
⇒ variants: dill, cloudpickle
joblib:

- **Simple parallel syntax:**
  
  ```python
  Parallel(n_jobs=2)(delayed(sqrt)(i) for i in range(10))
  ```

- **Fast persistence:**
  
  ```python
  joblib.dump(anything, 'filename.pkl.gz')
  ```

- **Primitive for out of core:**
  
  ```python
  pointer = mem.cache(f).call_and_shelves(big_data)
  ```

- Non-invasive syntax / paradigm
- Fast on big numpy arrays
- Soon backend system (job broker and persistence)

Gets job management into algorithms (eg in scikit-learn)
The Python VM is great

The simplicity of the VM is our strength

- Software Transactional Memory... would be nice
  But, I want to use foreign memory
  Java gained jmalloc for foreign memory

- Better garbage collection
  Yes but, I easily plug into reference counting

A strength of Python is its clear C API
  ⇒ Easy foreign functionality
The Python VM is great

The simplicity of the VM is our strength

Cython: the best of C and Python

- Add types for speed (numpy arrays as float*)
- Call C to bind external libraries: surprisingly easy

An adaptation layer between Python VM and C

A strength of Python is its clear C API

⇒ Easy foreign functionality
Working together
Scikit-learn is easy machine learning

As easy as py

```python
from sklearn import svm
classifier = svm.SVC()
classifier.fit(X_train, Y_train)
Y_test = classifier.predict(X_test)
```

People love the encapsulation

- The power of a simple object-oriented API
- Documentation-driven development
Scikit-learn is easy machine learning

As easy as py

```python
from sklearn import svm
classifier = svm.SVC()
classifier.fit(X_train, Y_train)
Y_test = classifier.predict(X_test)
```

People love the encapsulation
classifier is a semi black box

- The power of a simple object-oriented API
- Documentation-driven development

High-level, readable, simple API reduces cognitive load
PyData loves Python in return
Difference is richness

We all do different things

We can all benefit from others

though we don’t know how
Difference is richness, but requires outreach

We all do different things

We can all benefit from others

though we don’t know how

Being didactic outside one’s community is crucial

- Avoiding jargon
- Prioritizing information
- “Simple is better than complex”

Students learning numerics don’t care about unicode

Build documentation upon very simple examples

Think stackoverflow

Sphinx + Sphinx-gallery
Scientist ❤️ web dev: Python is the language for data

- Python language & VM is perfect to manipulate **low-level constructs**
  - with **high-level wordings**
  - Connects to other paradigms, eg C

@GaelVaroquaux
Scientist ❤️ web dev: Python is the language for data

- Python language & VM is perfect to manipulate **low-level constructs** with **high-level wordings**

- Dynamism and reflexivity
  ⇒ meta-programming and debugging
Scientist ✨ web dev: Python is the language for data

- Python language & VM is perfect to manipulate **low-level constructs** with **high-level wordings**

- Dynamism and reflexivity
  ⇒ **meta-programming and debugging**

- Needs for **compilation** and **dynamism**: a difficult balance
  PEP 509: guards on run-time modification
  PEP 510: function specialization

@GaelVaroquaux
Scientist ❤️ web dev: Python is the language for data

- Python language & VM is perfect to manipulate low-level constructs with high-level wordings
- Dynamism and reflexivity ⇒ meta-programming and debugging
- Needs for compilation and dynamism
- Pydata will use DB and concurrency from web
- PyData can give knowledge engineering + AI