UE 803 - Data Science

Session 1: setting up our working environment

Yannick Parmentier - Université de Lorraine / LORIA

Loria
Laboratoire lorrain de recherche
en informatique et ses applications

cnrs

UNIVERSITÉ
DE LORRAINE

Introduction

What is data science ?

Introduction

What is data science ?

> Interdisciplinary field that uses scientific *methods*, *algorithms* and *systems* to **extract knowledge** and insights **from data** in various forms.

Source: Wikipedia

Introduction

What is data science ?

> Interdisciplinary field that uses scientific *methods*, *algorithms* and *systems* to **extract knowledge** and insights **from data** in various forms.

Source: Wikipedia

Origins :

- close relation to *statistics* ("data analysis" field created by US mathematician John Tukey in 1962)
- 2002 : launching of the *Data Science Journal*

Introduction (continued)

What happened in the early 2000s ?

Introduction (continued)

What happened in the early 2000s ?

- a **cultural** evolution

> There are two cultures in the use of statistical modeling to reach conclusions from data. One assumes that the data are generated by a given stochastic data model. The other uses **algorithmic models** and treats the **data mechanism as unknown**.

(Leo Breitman, 2001)

Introduction (continued)

What happened in the early 2000s ?

- a **cultural** evolution

> There are two cultures in the use of statistical modeling to reach conclusions from data. One assumes that the data are generated by a given stochastic data model. The other uses **algorithmic models** and treats the **data mechanism as unknown**.

(Leo Breitman, 2001)

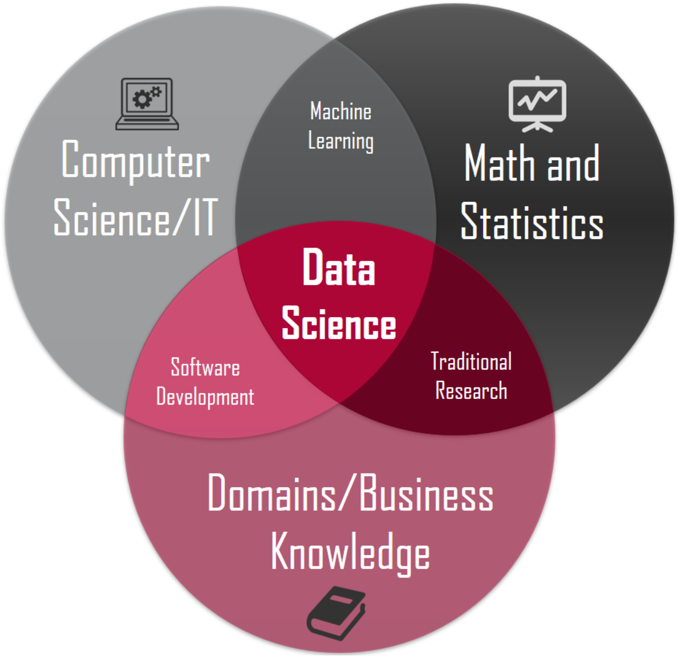
- a **technical** evolution
    - Big Data
    - Internet of Things

Introduction (continued)

In other terms :

> Data Science is about drawing useful conclusions from large and diverse data sets through **exploration**, **prediction**, and **inference**.

Source: inferentialthinking.com



Statistics : finding **patterns**

IT : **scaling up**, making predictions

Domain : **interpreting** results

Introduction (continued)

Classical workflow :

1) **Collect data**

2) **Clean and visualize data**

3) **Extract** underlying **knowledge** (data *features*)

4) **Apply Machine Learning** algorithms (e.g. linear regression, clustering, etc.) to make predictions

5) **Evaluate** the quality of the prediction

Course organization

- 60-hour course splitted into 3-hour sessions
- Each session includes practical exercises
- Evaluation is based on :
    - 3 reports on specific practical sessions
    - A final project (source code + oral defense)
- Implementation is done using the Python 3 programming language

Today's menu

- Setting up our working environment
- Hands on Python (again)
  - build-in data-types
  - Object-Oriented Programming

Data science with Python

Note it could have been R !

The Python ecosystem

- **Python** is an *open*, *interpreted*, *high-level*, *general-purpose*, *multi-paradigm*, *multi-platform* and *modular* programming language (Source: Wikipedia)

The Python ecosystem

- **Python** is an *open*, *interpreted*, *high-level*, *general-purpose*, *multi-paradigm*, *multi-platform* and *modular* programming language (Source: Wikipedia)

- **Python** is *actively maintained* by a large community (many 3rd-party *libraries*)

The Python ecosystem

- **Python** is an *open*, *interpreted*, *high-level*, *general-purpose*, *multi-paradigm*, *multi-platform* and *modular* programming language (Source: Wikipedia)

- **Python** is *actively maintained* by a large community (many 3rd-party *libraries*)

- Unlike e.g. Java, **Python** is *not backward compatible*

The Python ecosystem

- **Python** is an *open*, *interpreted*, *high-level*, *general-purpose*, *multi-paradigm*, *multi-platform* and *modular* programming language (Source: Wikipedia)

- **Python** is *actively maintained* by a large community (many 3rd-party *libraries*)

- Unlike e.g. Java, **Python** is *not backward compatible*

- Concretely, running a **Python** program requires:
  - a *platform* (Operating System),
  - a *language version* (interpreter),
  - and *librairies* (with potential compatibility issues)

The Python ecosystem

- **Python** is an *open*, *interpreted*, *high-level*, *general-purpose*, *multi-paradigm*, *multi-platform* and *modular* programming language (Source: Wikipedia)

- **Python** is *actively maintained* by a large community (many 3rd-party *libraries*)

- Unlike e.g. Java, **Python** is *not backward compatible*

- Concretely, running a **Python** program requires:
  - a *platform* (Operating System),
  - a *language version* (interpreter),
  - and *librairies* (with potential compatibility issues)

→ How to deal with all these sources of **heterogeneity** ?

Setting up a Python environment

Setting up a Python environment

- Running various versions of libraries (or interpreters)
  → **virtual environments** (*aka venv*)

Setting up a Python environment

- Running various versions of libraries (or interpreters)
  → **virtual environments** (*aka venv*)

- Virtual environments are **isolated environments** (interpreter + dependencies) which can be **activated/deactivated** *on demand*

Setting up a Python environment

- Running various versions of libraries (or interpreters)
  → **virtual environments** (*aka venv*)

- Virtual environments are **isolated environments** (interpreter + dependencies) which can be **activated/deactivated** *on demand*

- Virtual environments can be set up easily by using an adequate **package manager** such as Miniconda 🔗

Setting up a Python environment

- Running various versions of libraries (or interpreters)
→ **virtual environments** (*aka venv*)

- Virtual environments are **isolated environments** (interpreter + dependencies) which can be **activated/deactivated** *on demand*

- Virtual environments can be set up easily by using an adequate **package manager** such as Miniconda 🔗

- Virtual environments can be **created** and **activated**/deactivated using dedicated **commands**:

```
$ conda create --name myenv python=3.7
$ conda activate myenv
```

Setting up a Python environment (continued)

- From the terminal, libraries can be **searched** and then **installed** as follows

```
(myenv) ...$ conda search library_name
(myenv) ...$ conda install library_name
```

- Alternatively, libraries may be installed within conda virtual environments using `pip`

```
(myenv) ...$ pip install <library>
```

- Libraries installed by `conda` can be enumerated via

```
(myenv) ...$ conda list
```

→ Want more conda commands ? see conda cheatsheet 🔗

Programming in Python

- Option #1: in an **interactive** interpreter

```
$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print(3)
3
>>>
```

Programming in Python

- Option #1: in an **interactive** interpreter

```
$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print(3)
3
>>>
```

- Option #2: **in a source file**

```
python my_file.py
```

`my_file.py` can be edited by whatever editor you like : emacs, vim, geany, spyder, ...

Programming in Python (continued)

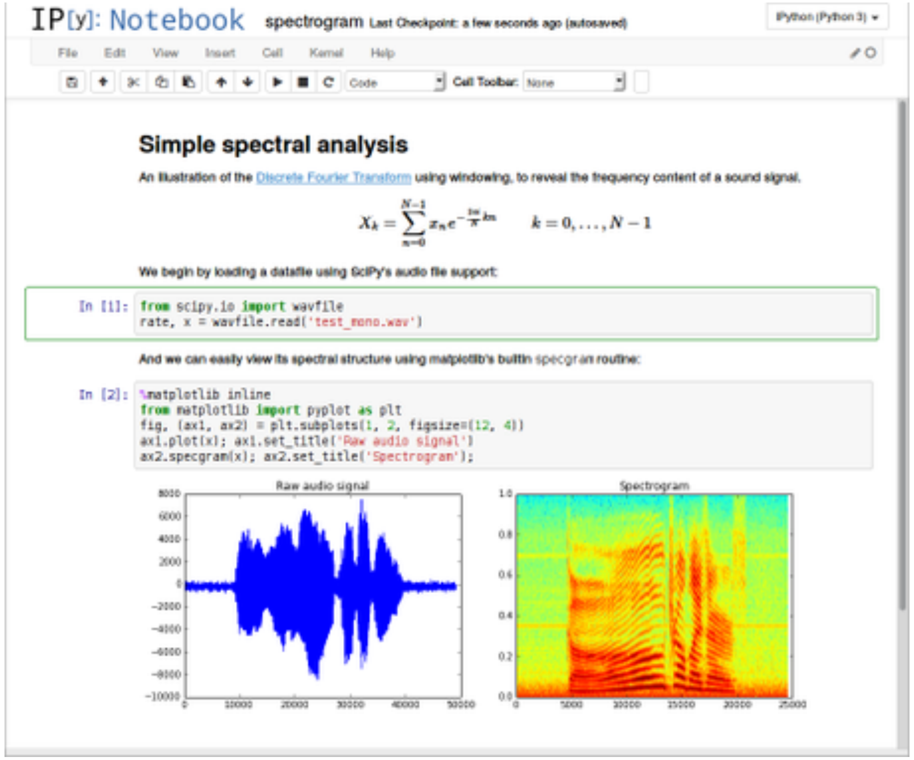
- Option #3: in an **integrated development environment** (IDE)
  - **edit *and* run** code from the same interface
- Example: Jupyter Notebook 🔗 (formerly iPython)
  - rich edition capacities **for code documentation** (e.g. formatted comments, images)
  - code and documentation are located **in a single file** (aka *literate programming*) called a **notebook**
  - Notebooks are made of blocks (aka *cells*) containing either **python code** or **markdown** + LaTeX **texts**

Programming in Python (continued)

- To **launch Jupyter from a terminal**:

```
(myenv) ...$ jupyter notebook
```

- Results :
  - web-browser displaying local files
  - notebooks corresponds to `.ipynb` files
  - hit `Ctrl+Shift` to interpret cells



Simple spectral analysis

An illustration of the Discrete Fourier Transform using windowing, to reveal the frequency content of a sound signal.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

We begin by loading a datafile using SciPy's audio file support:

```
In [1]: from scipy.io import wavfile
        rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin specgram routine:

```
In [2]: %matplotlib inline
        from matplotlib import pyplot as plt
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
        ax1.plot(x); ax1.set_title('Raw audio signal')
        ax2.specgram(x); ax2.set_title('Spectrogram');
```

Working with Python 3

Recall

- Python is :

Recall

- Python is :
    - *interpreted*

Recall

- Python is :
    - *interpreted*
    - *typed*

Recall

- Python is :
  - *interpreted*
  - *typed*
  - *indentation-sensitive*

Recall

- Python is :
    - *interpreted*
    - *typed*
    - *indentation-sensitive*
    - a *pass-by-reference* language

Recall

- Python is :
    - *interpreted*
    - *typed*
    - *indentation-sensitive*
    - a *pass-by-reference* language
    - *mutli-paradigm*
        - imperative
        - functional
        - object-oriented

Recall (continued)

- Rich built-in types :
  - int, float
  - boolean
  - string
  - list
  - dictionaries
  - None
  - *etc.*

Recall (continued)

- Rich built-in types :
  - int, float
  - boolean
  - string
  - list
  - dictionaries
  - None
  - *etc.*

```
>>> "hello"
'hello'
>>> print("hello")
hello
>>> type("hello")
<class 'str'>
>>>
```

Recall (continued)

- Type checking (with precise error messages!)

```
>>> 1 + 2.3
3.3
>>> True + 2.3
3.3
>>> "hello" + 12
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
>>>
```

Recall (continued)

- expressive language
  - conditions, loops
  - (named / unnamed, first-class) functions
  - (heterogeneous) lists (aka dynamic arrays)
  - dictionaries (aka associative arrays)
  - (non-mutable) tuples
  - lazy iterators (cf yield)

Recall (continued)

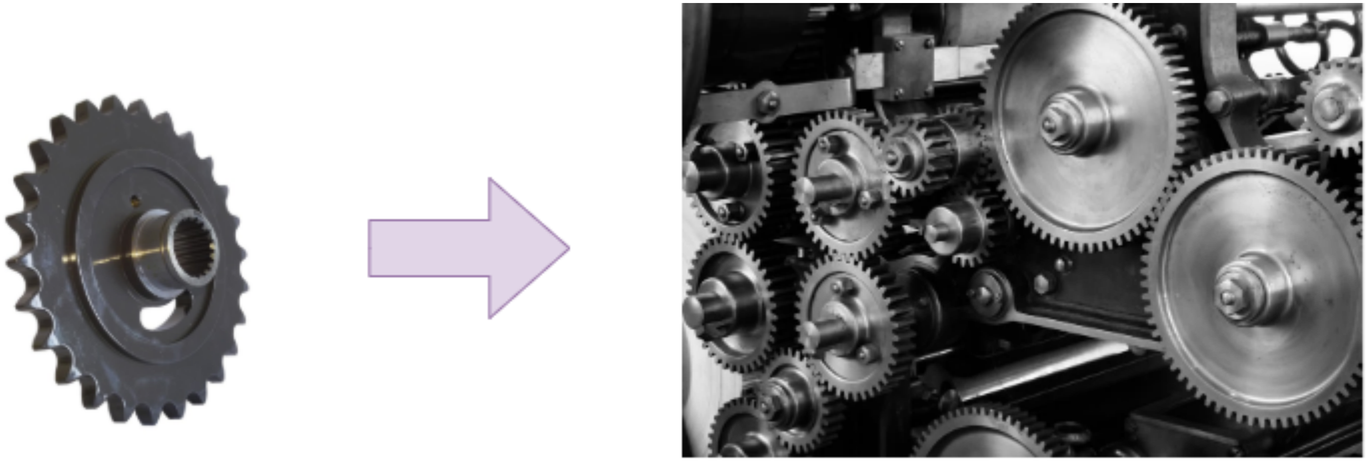
- and many very useful libraries
  - SCIentific PYthon toolkits (SciPy)
  - NL ToolKit (NLTK)
  - plotting (e.g. matplotlib)
  - window making (e.g TkInter, appJar)
  - game development (e.g pygame)
  - and much more

See also How to think like a computer scientist : Interactive Edition 🔗

Object-oriented Programming

From functional python

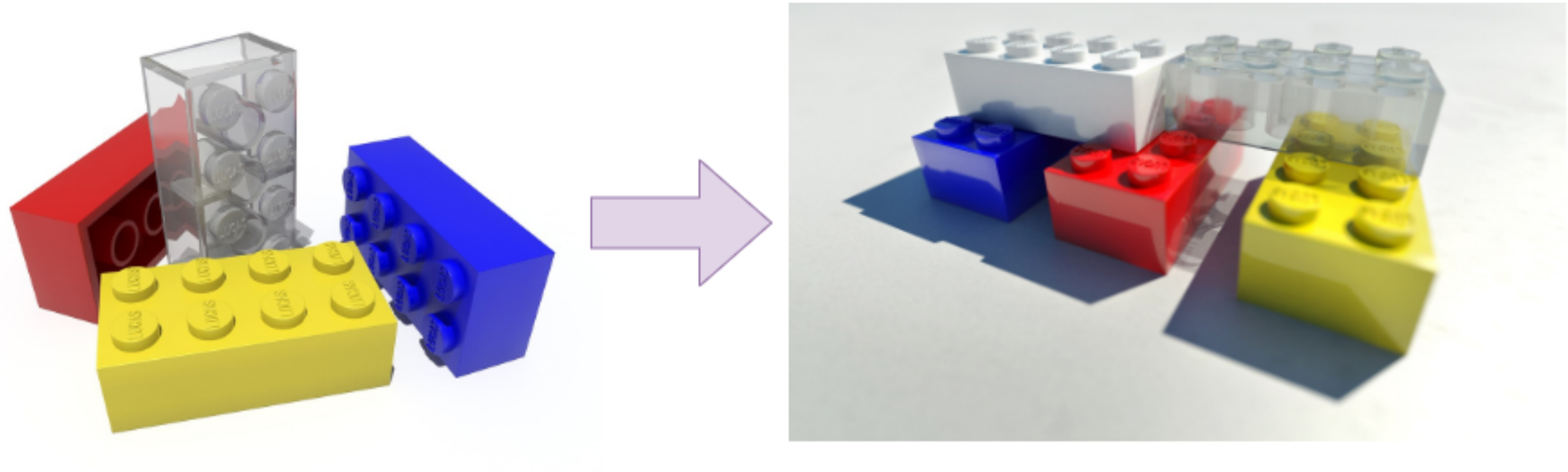
Focus on data *processings*



```
y = len(list(map(lambda x : x + 1, [1,2,3,4])))
```

To object-oriented python

Focus on data *types*



```
l = [1,2,3,4]
l.reverse()
```

Object-oriented programming

- *Data types* corresponds to **classes**
- *Classes* are made of **attributes** (variables) and **methods** (functions)
- A given data *instance* is called an **object**
- *Classes* can be defined locally or in **modules** (files)
- A program (python *main* script) can instantiate classes defined in accessible modules

Example

In a file named e.g. `Foo.py`, let us define a class `Foo` made of an attribute `bar` and three methods `__init__`, `__str__` and `concat`:

```python
class Foo:

  def __init__(self):
    self.bar = 'toto'

  def __str__(self):
    return self.bar + '!'

  def concat(self, s):
    self.bar += s
```

Example (continued)

Let us instantiate this class (object `x`) directly in `Foo.py`:

```python
if __name__=='__main__':
  x = Foo()
  print(x)
  x.concat('tata')
  print(x)
```

Finally, let us run this program by invoking:

```
python3 Foo.py
```

We will get:

```
toto!
tototata!
```

Sharing code

Sharing code

Many ways to share documents :

- USB sticks
- Emails
- Cloud-hosted drives (owncloud, dropbox, google drive, microsoft one, ...)

Sharing code

Many ways to share documents :

- USB sticks
- Emails
- Cloud-hosted drives (owncloud, dropbox, google drive, microsoft one, ...)

How to keep track of modifications ?

- complex file naming schemes

Sharing code

Many ways to share documents :

- USB sticks
- Emails
- Cloud-hosted drives (owncloud, dropbox, google drive, microsoft one, ...)

How to keep track of modifications ?

- complex file naming schemes

How to publish modifications ?

Introducing versioning

- A long history of **Version Control Systems** (CVS, SVN, Git, Darcs, Mercurial, ...)
- Versioning and sharing documents → **Development forge** (gitlab, github, bitbucket ...)
- Two main families :
    - centralized vs decentralized VCS
- Pros and cons ?
    - the cathedral and the bazaar

Introducing versioning (continued)

Versioning

- **keeps track of [recorded] modifications** (so-called `commits`, rolling back)
- **allows for experimentations** (within so-called `branches`, ~ temporary copies)

Introducing versioning (continued)

Versioning

- **keeps track of [recorded] modifications** (so-called `commits`, rolling back)
- **allows for experimentations** (within so-called `branches`, ~ temporary copies)

Together with a web hosting service (forge)

- **facilitates** team working

Introducing `git`

- three spaces to deal with :
  - your **working copy** of the project (obtained via either `git init` or `git clone`)
  - your **index** (space containing the current unrecorded modifications, located in `.git` subdir)
  - your **history** (space containing the code branches and versions, located in `.git` subdir)
- Each recorded modification is given an **identifier** (hash code)
- `HEAD` is the nickname of the last recorded modification

Introducing `git` (continued)

- Creating a new local project

  ```
  $ git init
  ```

- Getting information about local files

  ```
  $ git status
  ```

- Adding modifications to the index

  ```
  $ git add
  ```

- Recording modifications to the history

  ```
  $ git commit
  ```

Introducing `git` (continued)

- Creating a local clone of an existing project

  ```
  $ git clone <URL>
  ```

  (this existing project becomes the default *remote*)

- Pushing modifications to a remote

  ```
  $ git push [remote / branch]
  ```

- Pulling modifications from a remote

  ```
  $ git pull [remote / branch]
  ```

- Adding a remote (must have a common history!)

  ```
  $ git remote add <name> <URL>
  ```

Anatomy of git logs

```
git log
```

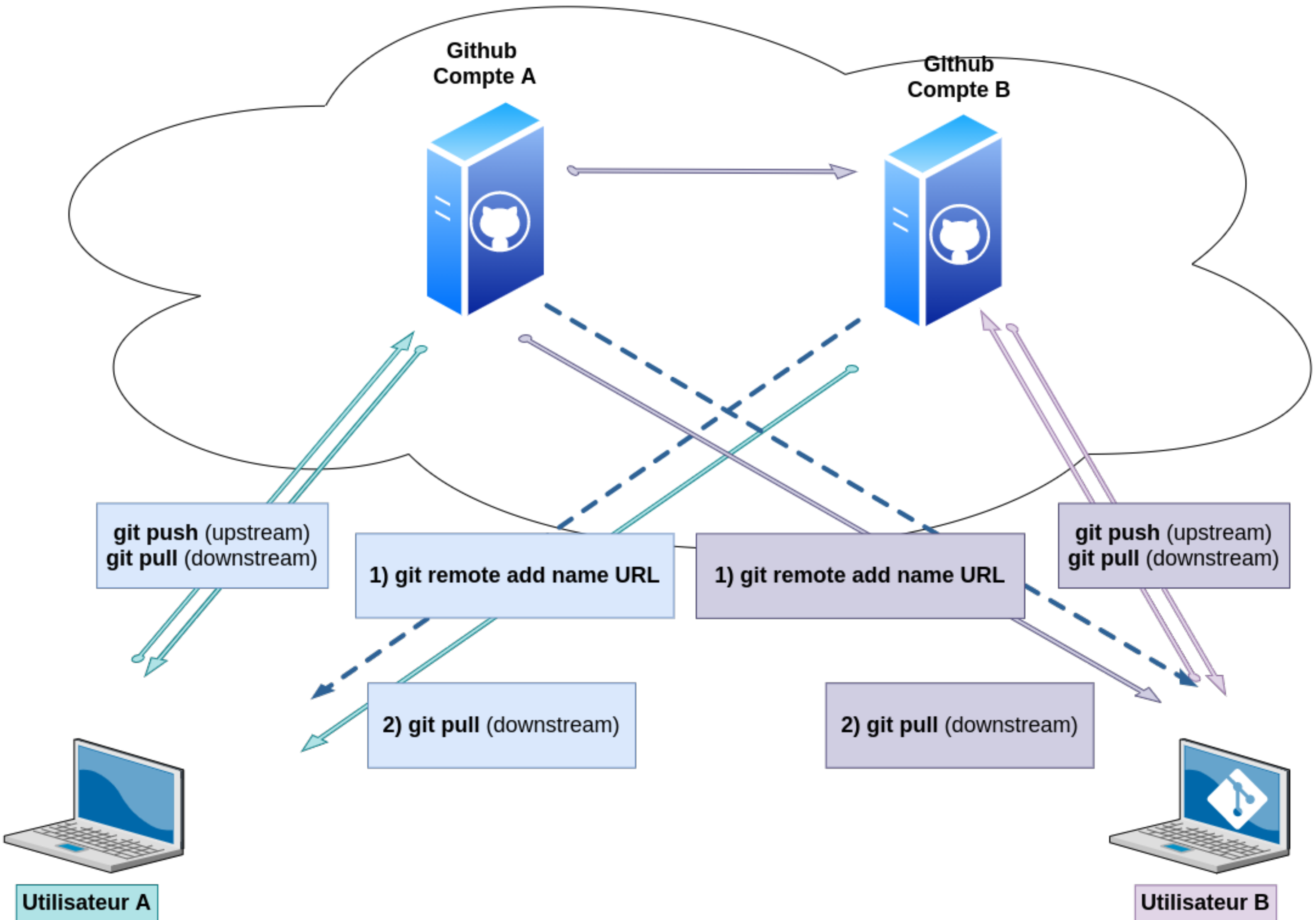
```
commit 2592da4330b4df6d482a631f4a35543b96f4744d
Merge: bff46dc 50b5135
Author: Alexander Matthes <ziz@mailbox.org>
Date:   Thu May 23 10:07:29 2019 +0200

    Merge branch 'master' of github.com:theZiz/aha

commit bff46dc3938df4699d92dc4a98cd57f8f2541448
Author: Alexander Matthes <ziz@mailbox.org>
Date:   Thu May 23 10:06:17 2019 +0200

    Added optional language attribute
```

Team work using `git`



git push (upstream)
git pull (downstream)

1) git remote add name URL

2) git pull (downstream)

1) git remote add name URL

2) git pull (downstream)

git push (upstream)
git pull (downstream)

For more information :

A visual Git Guide

Learn Git Branching

For future practical sessions

- **Get a copy of the notebook** used for the practical session (see the course on Arche/Moodle)

- **Run** you local notebook :

```
cd <where the notebook has been saved>
jupyter notebook
```

- Should you need to *locally* **keep track** of the versions of your work (global commands are invoked once on a given machine):

```
git config --global user.name "Your Name"
git config --global user.email "youremail@yourdomain.com"
git init .
git add <notebook_file>
git commit -m "<log message>"
```

Thank you!

Slideshow created using remark.

Exercise sheets (see the course on Arche/Moodle)

- Exercise sheet 1: Hands-on functional python
- Exercise sheet 2: Hands-on object-oriented python