Modèle de livre avec Jupyter Book

Lilian Besson

févr. 09, 2021

Introduction

Un modèle de livre créé avec Jupyter Book, en français et avec des exemples complets d'utilisations des extensions Sphinx, Jupyter et Jupyter Book les plus utiles.

Avertissement : Work in progress. . .

==> Voir https://perso.crans.org/besson/Info-Prepas-MP2I/Modele-de-livre-avec-Jupyter-Book/ pour la version actuelle de ce livre.

CHAPITRE 1

Content in Jupyter Book

There are many ways to write content in Jupyter Book. This short section covers a few tips for how to do so.

CHAPITRE 2

Welcome to your Jupyter Book

This is a small sample book to give you a feel for how book content is structured.

Check out the content pages bundled with this sample book to get started.

CHAPITRE 3

Markdown Files

Whether you write your book's content in Jupyter Notebooks (`.ipynb`) or in regular markdown files (`.md`), you'll write in the same flavor of markdown called **MyST Markdown**.

3.1 What is MyST ?

MyST stands for « Markedly Structured Text ». It is a slight variation on a flavor of markdown called « CommonMark » markdown, with small syntax extensions to allow you to write **roles** and **directives** in the Sphinx ecosystem.

3.2 What are roles and directives ?

Roles and directives are two of the most powerful tools in Jupyter Book. They are kind of like functions, but written in a markup language. They both serve a similar purpose, but **roles are written in one line**, whereas **directives span many lines**. They both accept different kinds of inputs, and what they do with those inputs depends on the specific role or directive that is being called.

3.2.1 Using a directive

At its simplest, you can insert a directive into your book's content like so :

```
```{mydirectivename}
My directive content
```
```

This will only work if a directive with name `mydirectivename` already exists (which it doesn't). There are many pre-defined directives associated with Jupyter Book. For example, to insert a note box into your content, you can use the following directive :

```
```{note}
Here is a note
```
```

This results in :

Note : Here is a note

In your built book.

For more information on writing directives, see the MyST documentation.

3.2.2 Using a role

Roles are very similar to directives, but they are less-complex and written entirely on one line. You can insert a role into your book's content with this pattern :

```
Some content {rolename}`and here is my role's content!`
```

Again, roles will only work if `rolename` is a valid role's name. For example, the `doc` role can be used to refer to another page in your book. You can refer directly to another page by its relative path. For example, the role syntax `{doc}`intro`` will result in : *Welcome to your Jupyter Book*.

For more information on writing roles, see the MyST documentation.

3.2.3 Adding a citation

You can also cite references that are stored in a `bibtex` file. For example, the following syntax : `{cite}`perez2011python`` will render like this : [PGH11].

Moreover, you can insert a bibliography into your page with this syntax. The `{bibliography}` directive must be used for all the `{cite}` roles to render properly. For example, if the references for your book are stored in `references.bib`, then the bibliography is inserted with :

```
```{bibliography} references.bib
```
```

Resulting in a rendered bibliography that looks like :

3.2.4 Executing code in your markdown files

If you'd like to include computational content inside these markdown files, you can use MyST Markdown to define cells that will be executed when your book is built. Jupyter Book uses *jupytext* to do this.

First, add Jupytext metadata to the file. For example, to add Jupytext metadata to this markdown page, run this command :

```
jupyter-book myst init --kernel python3 markdown.md
```

Once a markdown file has Jupytext metadata in it, you can add the following directive to run the code at build time :

```
```{code-cell}
print("Here is some code to execute")
```
```

When your book is built, the contents of any `{code-cell}` blocks will be executed with your default Jupyter kernel, and their outputs will be displayed in-line with the rest of your content.

```
print("Here is some code to execute")
```

```
Here is some code to execute
```

For more information about executing computational content with Jupyter Book, see The MyST-NB documentation.

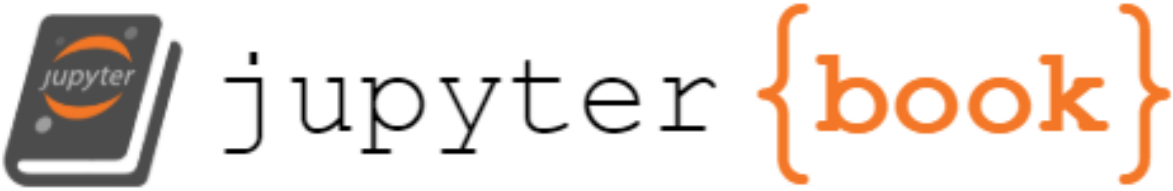
CHAPITRE 4

Content with notebooks

You can also create content with Jupyter Notebooks. This means that you can include code blocks and their outputs in your book.

4.1 Markdown + notebooks

As it is markdown, you can embed images, HTML, etc into your posts !



You an also add_{math} and

$$math^{blocks}$$

or

$$\text{mean} la_{tex}$$

$$mathblocks$$

$$\exists ? a \in \mathbb{R} \text{ tel que } \int_0^a \cos(x) \mathrm{d}x = 0$$

But make sure you $Escape $your $dollar signs $you want to keep !

4.2 MyST markdown

MyST markdown works in Jupyter Notebooks as well. For more information about MyST markdown, check out the MyST guide in Jupyter Book, or see the MyST markdown documentation.

4.3 Code blocks and outputs

Jupyter Book will also embed your code blocks and output in your book. For example, here's some sample Matplotlib code :

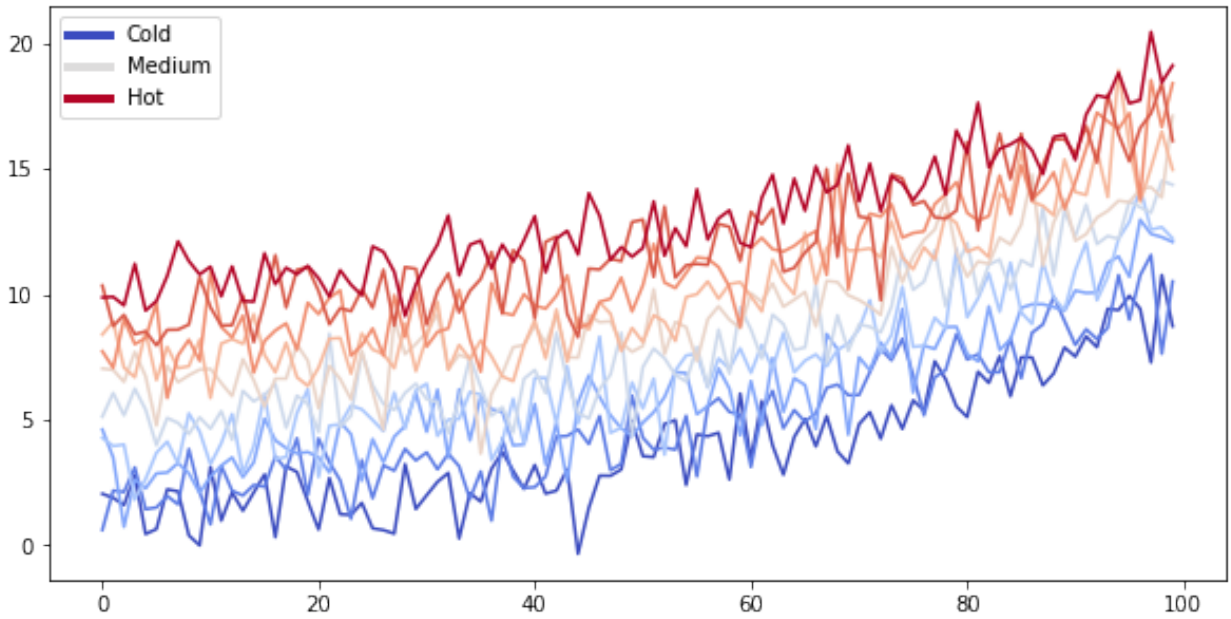
```
from matplotlib import rcParams, cycler
import matplotlib.pyplot as plt
import numpy as np
plt.ion()
```

```
# Fixing random state for reproducibility
np.random.seed(19680801)

N = 10
data = [np.logspace(0, 1, 100) + np.random.randn(100) + ii for ii in range(N)]
data = np.array(data).T
cmap = plt.cm.coolwarm
rcParams['axes.prop_cycle'] = cycler(color=cmap(np.linspace(0, 1, N)))


from matplotlib.lines import Line2D
custom_lines = [Line2D([0], [0], color=cmap(0.), lw=4),
                Line2D([0], [0], color=cmap(.5), lw=4),
                Line2D([0], [0], color=cmap(1.), lw=4)]

fig, ax = plt.subplots(figsize=(10, 5))
lines = ax.plot(data)
ax.legend(custom_lines, ['Cold', 'Medium', 'Hot']);
```



There is a lot more that you can do with outputs (such as including interactive outputs) with your book. For more information about this, see the Jupyter Book documentation.

CHAPITRE 5

Exemple de notebook avec Python

5.1 Explications

Le kernel Python est installé par défaut avec Jupyter.

5.2 Exemples

```
print("Ceci est du code Python")
```

```
Ceci est du code Python
```

```
import sys
print(sys.version)
```

```
3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0]
```

D'autres exemples ?

TODO : plus tard !

5.3 Pour en apprendre plus

— Ce petit tutoriel : https ://perso.crans.org/besson/apprendre-python.fr.html (sous licence GPLv3) ;
— Ce WikiBooks : https ://fr.wikibooks.org/wiki/Programmation_Python (sous licence libre) ;
— Ces deux livres de Python au niveau lycée : https ://github.com/exo7math/python1-exo7 et https ://github.com/exo7math/python2-exo7 (sous licence Creative Commons).

CHAPITRE 6

Exemple de notebook avec OCaml

6.1 Explications

Le kernel OCaml n'est pas installé par défaut avec Jupyter.

Il faut installer OPAM, puis ocaml-jupyter.

6.2 Exemples

```
Sys.command "ocaml -version";;
```

```
The OCaml toplevel, version 4.05.0
```

```
- : int = 0
```

```
print_endline "Bonjour depuis OCaml !";;
```

```
Bonjour depuis OCaml !
```

```
- : unit = ()
```

6.2.1 Une fonction récursive

```
let rec fact (n: int) : int =
    match n with
    | 0 -> 1
    | n -> n * (fact (n-1))
;;
```

```
val fact : int -> int = <fun>
```

```
for i = 1 to 15 do
    print_endline (Printf.sprintf "fact(%.2i) = %i" i (fact i))
done;;
```

```
fact(01) = 1
fact(02) = 2
fact(03) = 6
fact(04) = 24
fact(05) = 120
fact(06) = 720
fact(07) = 5040
fact(08) = 40320
fact(09) = 362880
fact(10) = 3628800
fact(11) = 39916800
fact(12) = 479001600
fact(13) = 6227020800
fact(14) = 87178291200
fact(15) = 1307674368000
```

```
- : unit = ()
```

6.2.2 Un type non paramétrique récursif et un exemple :

```
type formulePropositionnelle =
    | Var of string
    | Non of formulePropositionnelle
    | Ou of (formulePropositionnelle * formulePropositionnelle)
    | Et of (formulePropositionnelle * formulePropositionnelle)
;;
```

```
type formulePropositionnelle =
    Var of string
  | Non of formulePropositionnelle
  | Ou of (formulePropositionnelle * formulePropositionnelle)
  | Et of (formulePropositionnelle * formulePropositionnelle)
```

```
let x = Var("x")
and y = Var("y")
and z = Var("z")
;;
```

```
val x : formulePropositionnelle = Var "x"
val y : formulePropositionnelle = Var "y"
val z : formulePropositionnelle = Var "z"
```

```
let p1 = Ou(x, y)
and p2 = Et(y, z)
and p3 = Non(x)
;;

let p4 = Et(p1, p2);;
let p5 = Ou(p3, p4);;
let p6 = Non(p5);;
```

```
val p1 : formulePropositionnelle = Ou (Var "x", Var "y")
val p2 : formulePropositionnelle = Et (Var "y", Var "z")
val p3 : formulePropositionnelle = Non (Var "x")
```

```
val p4 : formulePropositionnelle =
  Et (Ou (Var "x", Var "y"), Et (Var "y", Var "z"))
```

```
val p5 : formulePropositionnelle =
  Ou (Non (Var "x"), Et (Ou (Var "x", Var "y"), Et (Var "y", Var "z")))
```

```
val p6 : formulePropositionnelle =
  Non (Ou (Non (Var "x"), Et (Ou (Var "x", Var "y"), Et (Var "y", Var "z"))))
```

```
let rec taille (formule: formulePropositionnelle) : int =
    match formule with
    | Var _ -> 1
    | Non phi -> 1 + taille phi
    | Et (phi1, phi2) | Ou (phi1, phi2) -> 1 + (taille phi1) + (taille phi2)
    | _ -> 0
;;
```

```
File "[63]", line 6, characters 6-7:
Warning 11: this match case is unused.
```

```
val taille : formulePropositionnelle -> int = <fun>
```

```
taille x;;
taille y;;
taille z;;
```

```
- : int = 1
```

```
- : int = 1
```

```
- : int = 1
```

```
taille p1;;
taille p2;;
taille p3;;
```

```
- : int = 3
```

```
- : int = 3
```

```
- : int = 2
```

Avec une fonction récursive terminale :

```
let taille (formule: formulePropositionnelle) : int =
    let rec aux acc = function
        | Var _ -> 1
        | Non phi -> aux (acc+1) phi
        | Et (phi1, phi2) | Ou (phi1, phi2) -> aux (aux (acc + 1) phi1) phi2
    in aux 0 formule
;;
```

```
val taille : formulePropositionnelle -> int = <fun>
```

```
taille p4;;
taille p5;;
taille p6;;
```

```
- : int = 7
```

```
- : int = 10
```

```
- : int = 11
```

6.2.3 Un type paramétrique récursif et un exemple :

```
type 'a arbreBinaire = Feuille | Noeud of ('a arbreBinaire * 'a * 'a arbreBinaire);;
```

```
type 'a arbreBinaire =
    Feuille
  | Noeud of ('a arbreBinaire * 'a * 'a arbreBinaire)
```

```
Noeud(Feuille, 10, Feuille)
```

```
- : int arbreBinaire = Noeud (Feuille, 10, Feuille)
```

6.2.4 D'autres exemples ?

TODO : plus tard !

6.3 Pour en apprendre plus

— Ce petit tutoriel : https ://perso.crans.org/besson/apprendre-python.fr.html (sous licence GPLv3) ;
— Ce WikiBooks : https ://fr.wikibooks.org/wiki/Programmation_Python (sous licence libre) ;
— Ces deux livres de Python au niveau lycée : https ://github.com/exo7math/python1-exo7 et https ://github.com/exo7math/python2-exo7 (sous licence Creative Commons).

CHAPITRE 7

Exemple de notebook avec C

7.1 Explications

Le kernel C n'est pas installé par défaut avec Jupyter.

Il faut installer jupyter-c-kernel

7.2 Exemples

```c
#include <stdio.h>

int main(int argc, char* argv) {
    printf("\nHello world from C!");
}
```

7.2.1 D'autres exemples ?

TODO : plus tard !

7.3 Pour en apprendre plus

- Ce petit tutoriel : https ://perso.crans.org/besson/apprendre-c.fr.html (sous licence GPLv3) ;
- Ce WikiBooks : https ://fr.wikibooks.org/wiki/Programmation_C (sous licence libre) ;
- Ce cours ArcSys1 de l'ENS Rennes : https ://mquinson.frama.io/ensr-arcsys1/ (2020-2021).

CHAPITRE 8

Exemple de notebook avec Bash

8.1 Explications

Voir https ://github.com/takluyver/bash_kernel

Pour installer :

```
$ pip install bash_kernel
$ python -m bash_kernel.install
```

À faire : Faire mieux ?

8.2 Exemples

La plupart des programmes en ligne de commande peuvent montrer leur version avec `programme --version` :

```
bash --version
```

```
GNU bash, version 4.4.20(1)-release (x86_64-pc-linux-gnu)
Copyright (C) 2016 Free Software Foundation, Inc.
Licence GPLv3+ : GNU GPL version 3 ou ultérieure <http://gnu.org/licenses/gpl.html>

Ceci est un logiciel libre ; vous être libre de le modifier et de le redistribuer.
AUCUNE GARANTIE n'est fournie, dans les limites permises par la loi.
```

```
jupyter-book --version
```

```
Jupyter Book: 0.10.0
MyST-NB: 0.11.1
Sphinx Book Theme: 0.0.39
MyST-Parser: 0.13.3
Jupyter-Cache: 0.4.2
NbClient: 0.5.1
```

Les commandes de bases pour traverser des dossiers :

```
echo "Affiche répertoire courant :"
pwd
```

```
Affiche répertoire courant :
/home/lilian/publis/Info-Prepas-MP2I/Modele-de-livre-avec-Jupyter-Book.git/notebooks
```

```
echo "Affiche fichiers du répertoire courant :"
ls -larth
```

```
Affiche fichiers du répertoire courant :
total 20K
drwxr-xr-x 6 lilian lilian 4,0K févr.  8 23:01  ../
drwxr-xr-x 2 lilian lilian 4,0K févr.  8 23:02  .ipynb_checkpoints/
-rw-r--r-- 1 lilian lilian 5,1K févr.  8 23:06 'Exemple de notebook avec Bash.ipynb'
drwxr-xr-x 3 lilian lilian 4,0K févr.  8 23:06  ./
```

```
echo "Change de répertoire courant :"
cd .ipynb_checkpoints/
```

```
Change de répertoire courant :
Vous quittez le dossier '/home/lilian/publis/Info-Prepas-MP2I/Modele-de-livre-avec-
↪Jupyter-Book.git/notebooks'.
        Direction ==> .ipynb_checkpoints/
```

Avertissement : Cette commande montre plus que la commande par défaut, à cause de mes alias.

À faire : Faire en sorte de ne pas montrer ça ?

```
ls -larth ./Exemple*Bash*.ipynb
```

```
-rw-r--r-- 1 lilian lilian 7,1K févr.  8 23:08 './Exemple de notebook avec Bash-
↪checkpoint.ipynb'
```

```
grep -n "ls -larth" ./Exemple*Bash*.ipynb
```

```
161:    "ls -larth"
224:    "ls -larth ./Exemple*Bash*.ipynb"
242:      "\u001b[32m\u001b[K161\u001b[m\u001b[K\u001b[36m\u001b[K:\u001b[m\u001b[K␣
↪ \"\u001b[01;31m\u001b[Kls -larth\u001b[m\u001b[K\"\n"
247:    "grep -n \"ls -larth\" ./Exemple*Bash*.ipynb"
```

D'autres exemples ?

TODO : plus tard !

8.3 Pour en apprendre plus

— Ce super tutoriel : http ://nmesnier.free.fr/tips.html
— Ce WikiBooks : https ://fr.wikibooks.org/wiki/Programmation_Bash
— Si vous connaissez d'autres shell : https ://hyperpolyglot.org/unix-shells

CHAPITRE 9

Exemple de notebook avec Rust

9.1 Explications

Le kernel Rust n'est pas installé par défaut avec Jupyter.

Il faut installer evcxr et le kernel evcxr_jupyter.

9.2 Exemples

9.2.1 Un exemple écrit par un de mes anciens élèves

Cf. ce petit challenge arithmético-algorithmique.

```
:dep chrono = "0.4"
use chrono::TimeZone;
use chrono::Datelike;

fn main() {
    let n = (1994..=2077)
        .filter(|n| chrono::Utc.ymd_opt(*n, 2, 29) == chrono::LocalResult::None)
        .map(|n| chrono::Utc.ymd(n, 2, 1))
        .filter(|t| t.weekday() == chrono::Weekday::Mon)
        .count();

    println!("{}", n);
}
```

```
main()
```

```
9
```

```
()
```

9.2.2 D'autres exemples ?

TODO : plus tard !

Avertissement : Je ne suis pas encore très expérimenté en Rust.

9.3 Pour en apprendre plus

— Ce petit tutoriel pour installer Rust : https ://rustup.rs/ ;
— https ://fasterthanli.me/articles/a-half-hour-to-learn-rust ;
— Ce livre : https ://doc.rust-lang.org/stable/book/

CHAPITRE 10

Exemple de notebook avec Java

10.1 Explications

Le kernel Java n'est pas installé par défaut avec Jupyter.

Il faut installer IJava.

10.2 Exemples

10.2.1 Un exemple écrit par un de mes anciens élèves

Cf. ce petit challenge arithmético-algorithmique.

```java
// ceci est du code Java 9 et pas Python !
// On a besoin des dépendances suivantes :
import java.util.Calendar;           // pour Calendar.FEBRUARY, .YEAR, .MONDAY
import java.util.GregorianCalendar;  // pour
import java.util.stream.IntStream;   // pour cet IntStream

IntStream.rangeClosed(1994, 2077)
         //.parallel()  // ce .parallel() est inutile, il y a trop peu de valeurs
         .mapToObj(i -> new GregorianCalendar(i, Calendar.FEBRUARY, 1))
         .filter(calendar -> !calendar.isLeapYear(calendar.get(Calendar.YEAR)))
         .filter(calendar -> calendar.get(Calendar.DAY_OF_WEEK) == Calendar.MONDAY)
         .count();
```

```
9
```

10.2.2 D'autres exemples ?

TODO : plus tard !

TODO prendre quelques cellules de notebooks dans https ://perso.crans.org/besson/teach/INF1_L1_Rennes1_2020-21/

10.3 Pour en apprendre plus

— Ce WikiBooks : https ://fr.wikibooks.org/wiki/Programmation_Java (sous licence libre) ;

CHAPITRE 11

1 reStructuredText Demonstration

Avertissement : Not written by me (Lilian Besson)

I'm a **NOT** the writer of this document !

Voir aussi :

The rST CheatSheet on GitHub

11.1 1.1 Examples of Syntax Constructs

Author David Goodger

Address 123 Example Street Example, EX Canada A1B 2C3

Contact docutils-develop@lists.sourceforge.net

Authors Me ; Myself ; I

organization humankind

date $Date : 2012-01-03 19 :23 :53 +0000 (Tue, 03 Jan 2012) $

status This is a « work in progress »

revision $Revision : 7302 $

version 1

copyright This document has been placed in the public domain. You may do with it as you wish. You may copy, modify, redistribute, reattribute, sell, buy, rent, lease, destroy, or improve it, quote it at length, excerpt, incorporate, collate, fold, staple, or mutilate it, or do anything else to it that your or anyone else's heart desires.

field name This is a generic bibliographic field.

field name 2 Generic bibliographic fields may contain multiple body elements.

Like this.

Dedication For Docutils users & co-developers.

abstract This document is a demonstration of the reStructuredText markup language, containing examples of all basic reStructuredText constructs and many advanced constructs.

Table of Contents

11.1.1 1.1.1 Structural Elements

1.1.1.1 Section Title

That's it, the text just above this line.

1.1.1.2 Transitions

Here's a transition :

It divides the section.

11.1.2 1.1.2 Body Elements

1.1.2.1 Paragraphs

A paragraph.

1.1.2.1.1 Inline Markup

Paragraphs contain text and may contain inline markup : *emphasis*, **strong emphasis**, `inline literals`, standalone hyperlinks (http://www.python.org), external hyperlinks (Python [5]), internal cross-references (*example*), external hyperlinks with embedded URIs (Python web site), footnote references (manually numbered [1], anonymous auto-numbered [3], labeled auto-numbered [2], or symbolic* [0]†), citation references ([CIT2002]), substitution references (

), and inline hyperlink targets (see *Targets* below for a reference back to here). Character-level inline markup is also possible (although exceedingly ugly !) in *re*`Structured`*Text*. Problems are indicated by |**problematic**| text (generated by processing errors ; this one is intentional).

The default role for interpreted text is *Title Reference*. Here are some explicit interpreted text roles : a PEP reference (**PEP 287**) ; an RFC reference (**RFC 2822**) ; a $_{\text{subscript}}$; a $^{\text{superscript}}$; and explicit roles for *standard* **inline** `markup`.

Let's test wrapping and whitespace significance in inline literals : `This is an example of --inline-literal --text, --including some-- strangely--hyphenated-words. Adjust-the-width-of-your-browser-window to see how the text is wrapped. -- ---- -------- Now note the spacing between the words of this sentence (words should be grouped in pairs).`

If the `--pep-references` option was supplied, there should be a live link to PEP 258 here.

5. http://www.python.org/

1. A footnote contains body elements, consistently indented by at least 3 spaces.

   This is the footnote's second paragraph.

3. This footnote is numbered automatically and anonymously using a label of « # » only.

2. Footnotes may be numbered, either manually (as in[1]) or automatically using a « # »-prefixed label. This footnote has a label so it can be referred to from multiple places, both as a footnote reference ([2]) and as a hyperlink reference (*label*).

0. Footnotes may also use symbols, specified with a « * » label. Here's a reference to the next footnote :†[0].

0. This footnote shows the next symbol in the sequence.

1.1.2.2 Bullet Lists

— A bullet list
  — Nested bullet list.
  — Nested item 2.
— Item 2.

  Paragraph 2 of item 2.
  — Nested bullet list.
  — Nested item 2.
    — Third level.
    — Item 2.
  — Nested item 3.

1.1.2.3 Enumerated Lists

1. Arabic numerals.
   a) lower alpha)
      (i) (lower roman)
          A. upper alpha.
          B. upper roman)
2. Lists that don't start at 1 :
   3. Three
   4. Four

   C. C
   D. D

   iii. iii
   iv. iv
3. List items may also be auto-enumerated.

1.1.2.4 Definition Lists

Term Definition

Term [classifier] Definition paragraph 1.
Definition paragraph 2.

Term Definition

1.1.2.5 Field Lists

what Field lists map field names to field bodies, like database records. They are often part of an extension syntax. They are an unambiguous variant of RFC 2822 fields.

how arg1 arg2 The field marker is a colon, the field name, and a colon.
The field body may contain one or more body elements, indented relative to the field marker.

1.1.2.6 Option Lists

For listing command-line options :

-a	command-line option « a »
-b file	options can have arguments and long descriptions
--long	options can be long also
--input=file	long options can also have arguments
--very-long-option	The description can also start on the next line. The description may contain multiple body elements, regardless of where it starts.
-x, -y, -z	Multiple options are an « option group ».
-v, --verbose	Commonly-seen : short & long options.
-1 file, --one=file, --two file	Multiple options with arguments.
/V	DOS/VMS-style options too

There must be at least two spaces between the option and the description.

1.1.2.7 Literal Blocks

Literal blocks are indicated with a double-colon (« : : ») at the end of the preceding paragraph (over there −−>). They can be indented :

```
if literal_block:
    text = 'is left as-is'
    spaces_and_linebreaks = 'are preserved'
    markup_processing = None
```

Or they can be quoted without indentation :

```
>> Great idea!
>
> Why didn't I think of that?
```

1.1.2.8 Line Blocks

This is a line block. It ends with a blank line.

Each new line begins with a vertical bar (« | »).

Line breaks and initial indents are preserved.

Continuation lines are wrapped portions of long lines ; they begin with a space in place of the vertical bar.

The left edge of a continuation line need not be aligned with the left edge of the text above it.

This is a second line block.

Blank lines are permitted internally, but they must begin with a « | ».

Take it away, Eric the Orchestra Leader !

A one, two, a one two three four

Half a bee, philosophically,

must, *ipso facto*, half not be.
But half the bee has got to be,
vis a vis its entity. D'you see?

But can a bee be said to be
or not to be an entire bee,
when half the bee is not a bee,
due to some ancient injury?

Singing...

1.1.2.9 Block Quotes

Block quotes consist of indented body elements :

> My theory by A. Elk. Brackets Miss, brackets. This theory goes as follows and begins now. All brontosauruses are thin at one end, much much thicker in the middle and then thin again at the far end. That is my theory, it is mine, and belongs to me and I own it, and what it is too.
>
> —Anne Elk (Miss)

1.1.2.10 Doctest Blocks

```
>>> print 'Python-specific usage examples; begun with ">>>"'
Python-specific usage examples; begun with ">>>"
>>> print '(cut and pasted from interactive Python sessions)'
(cut and pasted from interactive Python sessions)
```

1.1.2.11 Tables

Here's a grid table followed by a simple table :

Header row, column 1 (header rows optional)	Header 2	Header 3	Header 4
body row 1, column 1	column 2	column 3	column 4
body row 2	Cells may span columns.		
body row 3	Cells may span rows.	— Table cells — contain — body elements.	
body row 4			
body row 5	Cells may also be empty : −−>		

Inputs		Output
A	B	A or B
False	False	False
True	False	True
False	True	True
True	True	True

1.1.2.12 Footnotes

1.1.2.13 Citations

Here's a reference to the above, [CIT2002], and a [nonexistent] citation.

1.1.2.14 Targets

This paragraph is pointed to by the explicit « example » target. A reference can be found under *Inline Markup*, above. *Inline hyperlink targets* are also possible.

Section headers are implicit targets, referred to by name. See *Targets*, which is a subsection of *Body Elements*.

Explicit external targets are interpolated into references such as « Python[5] ».

Targets may be indirect and anonymous. Thus *this phrase* may also refer to the *Targets* section.

Here's a **`hyperlink reference without a target`_**, which generates an error.

1.1.2.14.1 Duplicate Target Names

Duplicate names in section headers or other implicit targets will generate « info » (level-1) system messages. Duplicate names in explicit targets will generate « warning » (level-2) system messages.

1.1.2.14.2 Duplicate Target Names

Since there are two « Duplicate Target Names » section headers, we cannot uniquely refer to either of them by name. If we try to (like this : **`Duplicate Target Names`_**), an error is generated.

1.1.2.15 Directives

These are just a sample of the many reStructuredText Directives. For others, please see http://docutils.sourceforge.net/docs/ref/rst/directives.html.

1.1.2.15.1 Document Parts

An example of the « contents » directive can be seen above this section (a local, untitled table of **contents_**) and at the beginning of the document (a document-wide *table of contents*).

1.1.2.15.2 Images

An image directive (also clickable – a hyperlink reference) :

A figure directive :

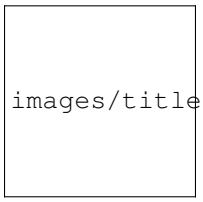


Fig. 11.1 – A figure is an image with a caption and/or a legend :

re	Revised, revisited, based on “re” module.
Structured	Structure-enhanced text, structuredtext.
Text	Well it is, isn’t it ?

This paragraph is also part of the legend.

1.1.2.15.3 Admonitions

Attention : Directives at large.

Prudence : Don’t take any wooden nickels.

Danger : Mad scientist at work !

Erreur : Does not compute.

Indication : It’s bigger than a bread box.

Important :
— Wash behind your ears.
— Clean up your room.
— Call your mother.
— Back up your data.

Note : This is a note.

Astuce : 15% if the service is good.

Avertissement : Strong prose may provoke extreme mental exertion. Reader discretion is strongly advised.

And, by the way…

You can make up your own admonition too.

1.1.2.15.4 Topics, Sidebars, and Rubrics

Sidebar Title

Optional Subtitle

This is a sidebar. It is for text outside the flow of the main text.

This is a rubric inside a sidebar

Sidebars often appears beside the main text with a border and background color.

Topic Title

This is a topic.

This is a rubric

1.1.2.15.5 Target Footnotes

1.1.2.15.6 Replacement Text

I recommend you try Python, *the* best language around[5].

1.1.2.15.7 Compound Paragraph

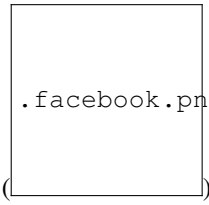
This paragraph contains a literal block :

```
Connecting... OK
Transmitting data... OK
Disconnecting... OK
```

and thus consists of a simple paragraph, a literal block, and another simple paragraph. Nonetheless it is semantically *one* paragraph.

This construct is called a *compound paragraph* and can be produced with the « compound » directive.

1.1.2.16 Substitution Definitions

An inline image (

) example :

(Substitution definitions are not visible in the HTML source.)

1.1.2.17 Comments

Here's one :

(View the HTML source to see the comment.)

11.1.3 1.1.3 Error Handling

Any errors caught during processing will generate system messages.

|* Expect 5 errors (including this one). ***|**

There should be six messages in the following, auto-generated section, « Docutils System Messages » :

CHAPITRE 12

MyST cheat sheet

Avertissement : Not written by me (Lilian Besson)

12.1 Headers

Syntax	Example	Note
`# Heading level 1` `## Heading level 2` `### Heading level 3` `#### Heading level 4` `##### Heading level 5` `###### Heading level 6`	`# MyST Cheat Sheet`	Level 1-6 headings, denoted by number of #

12.2 Target headers

Syntax	Example	Note
`(target_header)=`	`(myst_cheatsheet)=` `# MyST Cheat Sheet`	See *below* how to reference target headers.

12.2.1 Referencing target headers

Targets can be referenced with the ref inline role which uses the section title by default :

```
{ref}`myst_cheatsheet`
```

You can specify the text of the target :

```
{ref}`MyST syntax lecture <myst_cheatsheet>`
```

Another alternative is to use Markdown syntax :

```
[MyST syntax lecture](myst_cheatsheet)
```

12.3 Quote

Syntax	Example	Note
`> text`	`> this is a quote`	quoted text

12.4 Thematic break

Syntax	Example	Note
`---`	`This is the end of some text.` `---` `## New Header`	Creates a horizontal line in the output

12.5 Line comment

Syntax	Example	Note
`% text`	`a line` `% a comment` `another line`	See Comments for more information.

12.6 Block break

Syntax	Example	Result
`+++`	`This is an example of` `+++ {"meta": "data"}` `a block break`	This is an example of a block break

12.7 HTML block

Syntax	Example	Result
`<tagName> text <tagName>`	`<p> This is a␣` `↪paragraph </p>`	

12.8 Links

Syntax	Example	Result
`[text](target)`	`[Jupyter Book](https://` `↪jupyterbook.org)`	Jupyter Book
`[text](relative_path)`	`[PDF documentation](../advanced/` `↪pdf)`	PDF documentation
`[text](target "title")`	`[Jupyter Book](https://` `↪jupyterbook.org "JB Homepage")`	Jupyter Book
`<target>`	`<https://jupyterbook.org>`	https://jupyterbook.org
`[text][key]`	`[Jupyter Book][intro_page]` `[intro_page]: https://` `↪jupyterbook.org`	Jupyter Book

12.9 Lists

12.9.1 Ordered list

Example	Result
`1. First item` `2. Second item` `    1. First sub-item`	1. First item 2. Second item 1. First sub-item
`1. First item` `2. Second item` `    * First sub-item`	1. First item 2. Second item — First subitem

12.9.2 Unordered list

Example	Result
`* First item` `* Second item` `  * First subitem`	— First item — Second item — First subitem
`* First item` `  1. First subitem` `  2. Second subitem`	— First item 1. First subitem 2. Second subitem

12.10 Tables

Syntax	Example	Result
<pre>| a | b | | :--- | ---: | | c | d |</pre>	<pre>| Training | Validation | | :--- | ---: | | 0 | 5 | | 13720 | 2744 |</pre>		Training	Validation | 0	5 13720	2744
<pre>```{list-table} :header-rows: 1 * - Col1 - Col2 * - Row1 under Col1 - Row1 under Col2 * - Row2 under Col1 - Row2 under Col2 ```</pre>	<pre>```{list-table} :header-rows: 1 :name: example-table * - Training - Validation * - 0 - 5 * - 13720 - 2744 ```</pre>	Tableau 12.1 – My table title Training | Validation 0 | 5 13720 | 2744
<pre>```{list-table} Table title :header-rows: 1 * - Col1 - Col2 * - Row1 under Col1 - Row1 under Col2 * - Row2 under Col1 - Row2 under Col2 ```</pre>	<pre>```{list-table} This table title :header-rows: 1 * - Training - Validation * - 0 - 5 * - 13720 - 2744 ```</pre>	Tableau 12.2 – This table title Training | Validation 0 | 5 13720 | 2744

12.10.1 Referencing tables

Note : In order to reference a table you must add a label to it. To add a label to your table simply include a `:name:` parameter followed by the label of your table. In order to add a numbered reference, you must also include a table title. See example above.

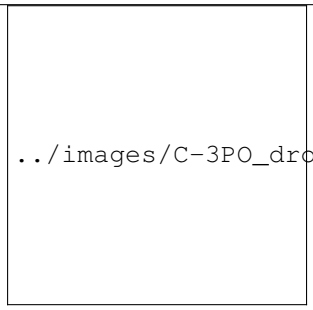
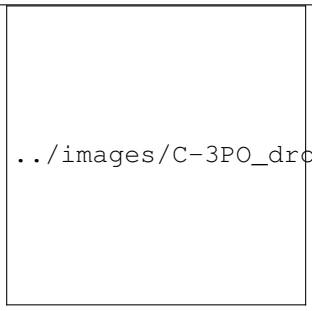
Syntax	Example	Result
`{numref}`label``	`{numref}`example-table` is an example.`	Tableau 12.1 is an example.
`{ref}`text <label>``	`This {ref}`table <example-table>` is an example.`	This *table* is an example.
`{numref}`text %s <label>``	`{numref}`Tbl %s <example-table>` is an example.`	Tbl 12.1 is an example.

12.11 Admonitions

Syntax	Example	Result
``{admonition} Title text ```	```{admonition} This is a title An example of an admonition with a title. ```	**This is a title** An example of an admonition with a title.
``{note} text ``` or ```{note} text some more text. .. ```	```{note} Notes require **no** arguments, so content can start here. ```	**Note :** Notes require **no** arguments, so content can start here.
``{warning} text some more text. .. ```	```{warning} This is an example of a warning directive. ```	**Avertissement :** This is an example of a warning directive.
``{tip} text some more text. .. ```	```{tip} This is an example of a tip directive. ```	**Astuce :** This is an example of a tip directive.
``{caution} text some more text. .. ```	```{caution} This is an example of a caution directive. ```	**Prudence :** This is an example of a caution directive.
``{attention} text some more text. ..	```{attention} This is an example of an attention directive. ```	**Attention :** This is an example of an attention directive.

12.12 Figures and images

Note : Content is not permitted in the image directive.

Syntax	Example	Result
`` ```{figure} ./ ↪path/to/ ↪figure.jpg :name: label caption ``` ````	```` ```{figure} ../images/C-3PO_ ↪droid.png :height: 150px :name: figure-example Here is my figure caption! ``` ````	 Fig. 12.1 – Here is my figure caption !
`` ```{image} ./ ↪path/to/ ↪figure.jpg :name: label ``` ````	```` ```{image} ../images/C-3PO_ ↪droid.png :height: 150px :name: image-example ``` ````	

See ../content/figures and ../file-types/markdown for more information.

12.12.1 Referencing figures

Syntax	Example	Result
`{numref}`label``	`{numref}`figure-example`is a figure example.`	Fig. 12.1 is a figure example.
`{numref}`text %s <label>``	`{numref}`Figure %s <figure-example>` is an example.`	Figure 12.1 is an example.
`{ref}`text <label>``	`This {ref}`figure <figure-example>` is an example.`	This *figure* is an example.

12.12.2 Referencing images

Syntax	Example	Result
`{ref}`text <label>``	`This {ref}`image <image-example>` is an example.`	This *image* is an example.

12.13 Math

Syntax	Example	Result
Inline	`This is an example of an inline equation $z=\sqrt{x^2+y^2}$.`	This is an example of an inline equation $z = \sqrt{x^2 + y^2}$.
Math blocks	`This is an example of a math block` `$$` `z=\sqrt{x^2+y^2}` `$$`	This is an example of a math block $$z = \sqrt{x^2 + y^2}$$
Math blocks with labels	`This is an example of a math block with a label` `$$` `z=\sqrt{x^2+y^2}` `$$ (mylabel)`	This is an example of a math block with a label $$z = \sqrt{x^2 + y^2} \quad (12.1)$$
Math directives	`This is an example of a math directive with a label` ```` ```{math} ```` `:label: eq-label` `z=\sqrt{x^2+y^2}` ```` ``` ````	This is an example of a math directive with a label $$z = \sqrt{x^2 + y^2} \quad (12.2)$$

See ../content/math for more information.

12.13.1 Referencing math directives

Syntax	Example	Result
`{eq}`label``	`Check out equation {eq}`eq-label`.`	Check out equation (12.2).

12.14 Code

12.14.1 In-line code

Example :

```
Wrap in-line code blocks in backticks: `boolean example = true;`.
```

Result :

Wrap in-line code blocks in backticks : `boolean example = true;`.

12.14.2 Code and syntax highlighting

Example :

````
```python
note = "Python syntax highlighting"
print(node)
```
````

or

````
```
No syntax highlighting if no language
is indicated.
```
````

Result :

```python
note = "Python syntax highlighting"
print(node)
```

or

```
No syntax highlighting if no language
is indicated.
```

12.14.3 Executable code

Avertissement : Make sure to include this front-matter YAML block at the beginning of your `.ipynb` or `.md` files.

```
---
jupytext:
  formats: md:myst
  text_representation:
    extension: .md
    format_name: myst
kernelspec:
  display_name: Python 3
  language: python
  name: python3
---
```

Example :

````
```{code-cell} ipython3
note = "Python syntax highlighting"
print(note)
```
````

Result :

```
note = "Python syntax highlighting"
print(note)
```

```
Python syntax highlighting
```

See ../file-types/myst-notebooks for more information.

Tags

The following `tags` can be applied to code cells by introducing them as options :

Tag option	Description	Example
`"full-width"`	Cell takes up all of the horizontal space	```` ```{code-cell} ipython3 :tags: ["full-width"] print("This is a test.") ``` ````
`"output_scroll"`	Make output cell scrollable	```` ```{code-cell} ipython3 :tags: ["output_scroll"] for ii in range(100): print("This is a test.") ``` ````
`"margin"`	Move code cell to the right margin	```` ```{code-cell} ipython3 :tags: ["margin"] print("This is a test.") ``` ````
`"hide-input"`	Hide cell but the display the outputs	```` ```{code-cell} ipython3 :tags: ["hide-input"] print("This is a test.") ``` ````
`"hide-output"`	Hide the outputs of a cell	```` ```{code-cell} ipython3 :tags: ["hide-output"] print("This is a test.") ``` ````
`"hide-cell"`	Hides inputs and outputs of code cell	```` ```{code-cell} ipython3 :tags: ["hide-cell"] print("This is a test.") ``` ````
`"remove-input"`	Remove the inputs of a cell	```` ```{code-cell} ipython3 :tags: ["remove-input"] print("This is a test.") ``` ````
`"remove-output"`	Remove the outputs of a cell	```` ```{code-cell} ipython3 :tags: ["remove-output"] print("This is a test.") ``` ````
`"remove-cell"`	Remove the entire code cell	```` ```{code-cell} ipython3 :tags: ["remove-cell"] print("This is a test.") ``` ````
`"raises-exception"`	Mark cell as « expected to error »	```` ```{code-cell} ipython3 :tags: ["raises-exception"] while True print('Hello world') ``` ````

12.14.4 Gluing variables

Example :

```
```{code-cell} ipython3
from myst_nb import glue
my_variable = "here is some text!"
glue("glued_text", my_variable)
```

Here is an example of how to glue text: {glue:}`glued_text`
```

Result :

```
from myst_nb import glue
my_variable = "here is some text!"
glue("glued_text", my_variable)
```

```
'here is some text!'
```

Here is an example of how to glue text : 'here is some text!'

See glue/gluing for more information.

12.14.5 Gluing numbers

Example :

```
```{code-cell} ipython3
from myst_nb import glue
import numpy as np
import pandas as pd

ss = pd.Series(np.random.randn(4))
ns = pd.Series(np.random.randn(100))

glue("ss_mean", ss.mean())
glue("ns_mean", ns.mean(), display=False)
```

Here is an example of how to glue numbers: {glue:}`ss_mean` and {glue:}`ns_mean`.
```

Result :

```
from myst_nb import glue
import numpy as np
import pandas as pd

ss = pd.Series(np.random.randn(4))
ns = pd.Series(np.random.randn(100))

glue("ss_mean", ss.mean())
glue("ns_mean", ns.mean(), display=False)
```

```
0.08106869626773033
```

Here is an example of how to glue numbers : `0.08106869626773033` and `-0.04866537730448551`.

See glue/gluing for more information.

12.14.6 Gluing visualizations

Example :

```
```{code-cell} ipython3
from myst_nb import glue
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 200)
y = np.sin(x)
fig, ax = plt.subplots()
ax.plot(x, y, 'b-', linewidth=2)

glue("glued_fig", fig, display=False)
```

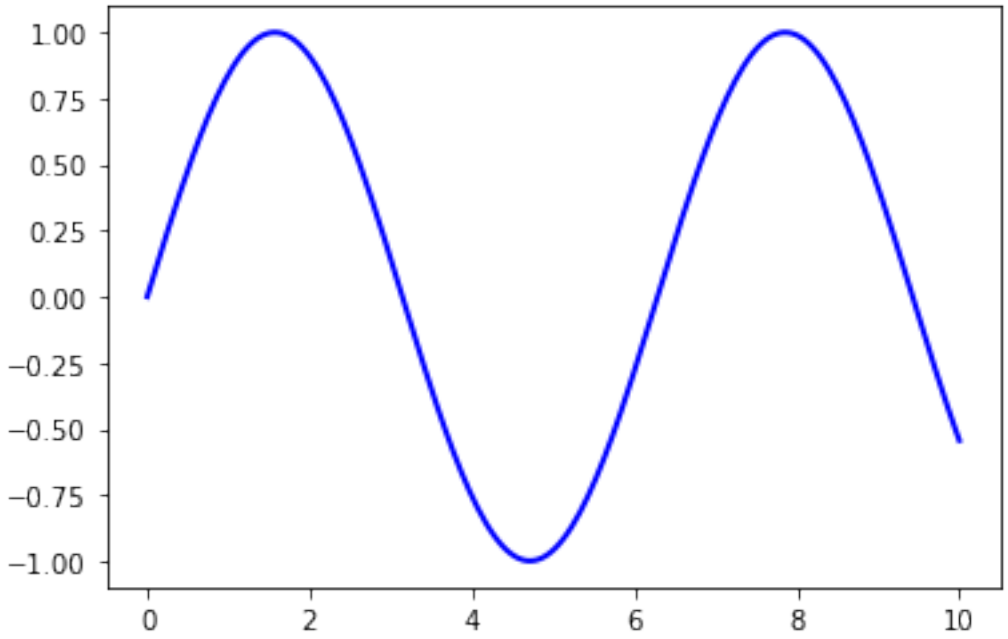
This is an inline glue example of a figure: {glue:}`glued_fig`.
This is an example of pasting a glued output as a block:
```{glue:} glued_fig
```
```

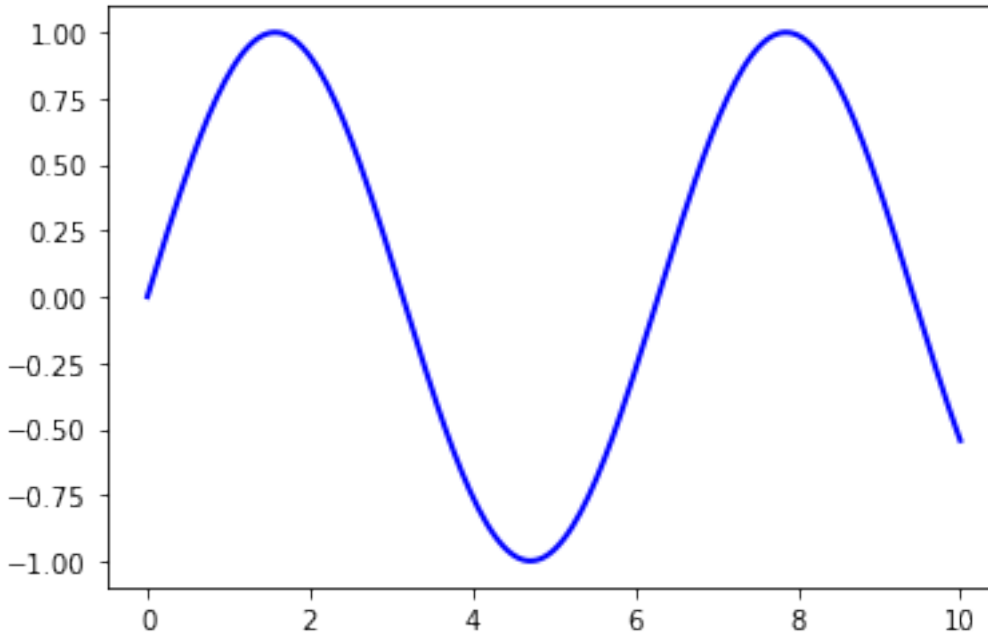
Result :

```
from myst_nb import glue
import matplotlib.pyplot as plt
import numpy as np

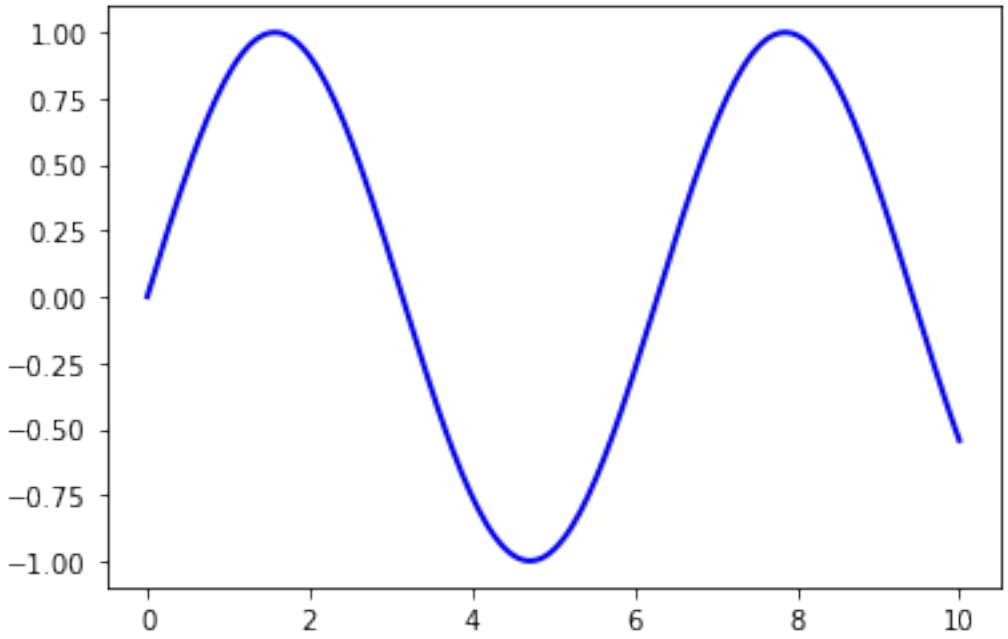
x = np.linspace(0, 10, 200)
y = np.sin(x)
fig, ax = plt.subplots()
ax.plot(x, y, 'b-', linewidth=2)

glue("glued_fig", fig, display=False)
```





This is an inline glue example of a figure :
This is an example of pasting a glued output as a block :



See glue/gluing for more information.

12.14.7 Gluing math

Example :

```
```{code-cell} ipython3
import sympy as sym
x, y = sym.symbols('x y')
z = sym.Function('z')
z = sym.sqrt(x**2+y**2)
glue("example_eq", z, display=False)
```

To glue a math equation try
```{glue:math} example_eq
:label: glue-eq-example
```
```

Result :

```
import sympy as sym
x, y = sym.symbols('x y')
z = sym.Function('z')
z = sym.sqrt(x**2+y**2)
glue("example_eq", z, display=False)
```

To glue a math equation try :

$$\sqrt{x^2 + y^2} \tag{12.3}$$

See for more information.

12.15 Reference documents

Syntax	Example	Result
`{doc}`path/to/document``	`See {doc}`../content/citations` for more information.`	See ../content/citations for more information.
`{doc}`text <path/to/document>``	`See {doc}`here <../content/citations>` for more information.`	See here for more information.

12.16 Footnotes

Note : Footnotes are displayed at the very bottom of the page.

Syntax	Example	Result
`[^ref]` `[^ref]: Footnote text`	`This is an example of a footnote.[^footnote1]` `[^footnote1]: The definition for referencing footnotes is generally placed at the bottom of the document.`	This is a footnote reference. [1]

See Footnotes for more information.

12.17 Citations

Note : Make sure you have a reference bibtex file. You can create one by running `touch references.bib` or view a `references.bib` example.

1. This **is** the footnote definition.

Syntax	Example	Result
{cite}`mybibtexcitation`	This example generates the following citation {cite}`perez2011python`.	This example generates the following citation [PGH11].

To include a list of citations mentioned in the document, introduce the bibliography directive

```
```{bibliography} references.bib
:filter: docname in docnames
```
```

See ../content/citations for more information.

CHAPITRE 13

Glossaire

Cette page est un glossaire, qui liste des termes et les définitions que j'en donne.

Avertissement : Ces définitions contiennent parfois mon propre point de vue, qui n'est en rien celui de mes employeurs, passés, actuels ou futurs.

13.1 Vocabulaire d'informatique

Algorithme Ce qui est étudié par l'algorithmique. TODO :

Algorithmique Science qui étudie les algorithmes. TODO :

Informatique Science de l'information, de ses traitements et manipulations, de ses représentations. L'informatique est à la fois une science (la branche des mathématiques qui répond notamment à la question « Qu'est-ce qu'un calcul ? »), une industrie, un rayon de supermarché, un ensemble d'outils techniques que tout le monde utilise… Autrement dit, l'informatique mélange des aspects théoriques, techniques, commerciaux.

Programme Produit par programmation. TODO :

Programmation Qui produit des programmes. TODO :

Bogue Version française d'un *bug* en anglais : un problème dans un programme informatique.

Bug On lui préférera la version française *bogue*.

Logiciel TODO :

Langage TODO :

Logiciel libre Free software : *Free as in free speech, not free beer !* TODO :

Open source Pas comme logiciel libre

Logiciel propriétaire L'inverse des logiciels libres.

13.2 Vocabulaire de la programmation

Signature TODO :

Variable TODO :

Liste TODO :

Tableau TODO :

Dictionnaire TODO :

Référence TODO :

Passage par valeur TODO :

Passage par référence TODO :

Passage par recopie TODO :

Tuple Aussi appelés couples ou n-uplet en français. TODO :

Classe TODO :

Objet TODO :

13.2.1 Vocabulaire spécifique à l'architecture des ordinateurs

Mémoire RAM/ROM/ ? TODO :

RAM TODO :

CPU Computation Processing Unit TODO :

Transistor TODO :

Mémoire cache TODO :

ALU TODO :

FLU TODO :

GPU Graphical Processing Unit. Utilisés pour vos jeux vidéo, mais aussi pour le calcul numérique intensif hautement parallèle, avec des langages comme nVidia CUDA, ou des outils automatiques comme Google Tensorflow ou Numba pour Python. TODO :

Cœur Un des CPU mono-cœur dans un CPU multi-cœur.

13.2.2 Matériels informatique

Écran TODO :

Clavier TODO :

Souris TODO :

Écran tactile TODO :

Ethernet TODO :

WiFi TODO :

Bluetooth TODO :

Webcam TODO :

Casque audio TODO :

Microphone TODO :

13.2.3 Vocabulaire spécifique à Internet

Protocole TODO :

TCP TODO :

UDP TODO :

IP TODO :

IPv6 TODO :

DNS TODO :

Adresse TODO :

URL TODO :

13.2.4 Verbes spécifiques en informatique

Coder Synonyme de programmer. J'aime bien, c'est rapide et très oral.

Implémenter Synonyme de programmer. J'aime bien, ça fait classe.

Implanter Synonyme de programmer. J'aime pas, on dirait qu'on plante des arbres. Moi je code des scripts, j'implémente des algorithmes, et je programme en général.

Programmer Synonyme de coder, implémenter, implanter.

Compiler Exécuter un compilateur. TODO :

Débogguer Version française de *to debug* en anglais : chasser les bogues d'un programme informatique.

Debug On lui préférera la version française *débogguer*.

Retourner Un anglicisme dans la plupart des cas, qui est à éviter : une fonction *renvoie* une valeur, elle ne la retourne pas... Sauf si la fonction demandée doit vraiment *retourner* une liste (`[x0,..,xN] -> [xN,..,x0]`) ou une chaîne (`"Canoë" -> "ëonaC`). Je serai impitoyable sur cet anglicisme !

Renvoyer Lorsque une fonction (ou un programme) calcule quelque chose et transmet le résultat à la fonction ou à l'environnement appelant, on doit dire qu'elle a *renvoyé* ce résultat, par *retourner*, qui est un anglicisme (*to return*).

13.3 Liste de langages

13.3.1 Langages utilisés en prépas MP2I

OCaml Langage de programmation créé par l'INRIA, principalement fonctionnel mais aussi impératif et orienté objet (*Objective Caml*). Disponible sur toutes les plateformes, gratuitement. Un des langages enseignés en classes préparatoires (avec *Python*, *SQL* et *C*). Cf. https://www.ocaml.org/.

Python Langage de programmation impératif, mais aussi orienté objet. Disponible sur toutes les plateformes, gratuitement. Un des langages enseignés en classes préparatoires (avec *SQL*, *C* et *OCaml*). Cf. https://www.python.org/, et WikiBooks Python ainsi que https://python-prepa.github.io/. Mes suggestions pour apprendre Python sont sur https://perso.crans.org/besson/apprendre-python.fr.html

SQL Langage de requête de bases de données (*Structured Query Language*). Disponible en plein de variantes, nous nous focaliserons sur SQLite. Disponible sur toutes les plateformes, gratuitement. Un des langages enseignés en classes préparatoires (avec *OCaml*, *Python* et *C*). Cf. https://sql.sh/ pour un très bon cours en français.

SQLite Un petit SQL, qui n'a pas besoin d'un serveur gestionnaire de base de données mais qui stocke la base de données sur un fichier local sur un disque. Pratique pour enseigner et pratiquer le SQL sans se prendre la tête avec un serveur, et pour de petites applications dont les performances n'importent peu et dont les bases de données sont petites. Cf. https://www.sqlite.org/.

C Langage de programmation impératif, très proche de la machine (bas niveau). Un des principaux langages utilisés au monde, quasiment disponible sur toutes les plateformes, gratuitement. Un des langages enseignés en classes préparatoires (avec *OCaml*, *Python* et *SQL*).

Bash Langage de script du projet GNU Bash, utilisé depuis les années 1980. Ce n'est pas un des langages enseignés en classes préparatoires mais nous nous en servirons quand même. Si vous devenez accro à GNU/Linux je vous suggère de travailler avec `Fish Shell <https://www.fishshell.com>_`, plus moderne et plus agréable à utiliser.

Makefile Un petit langage de description de tâches, généralement utilisé pour faciliter la compilation de code, très populaire depuis les années 1980. Ce n'est pas un des langages enseignés en classes préparatoires mais nous nous en servirons quand même. Les Makefiles sont notamment utiles pour écrire des programmes non triviaux en *C* et en *OCaml*. Ce livre est généré grâce à un *Makefile* TODO : lien.

13.3.2 D'autres langages

JavaScript Un autre langage très populaire, utilisé principalement pour programmer des petites fonctionnalités dans des pages web, mais aussi de plus en plus pour des applications mobiles ou bureaux (avec Electron) ou des applications en lignes de commande ou côté serveur (avec nodejs). Ces ressources peuvent vous aider à apprendre : WikiBooks JavaScript, https://javascript.info/ TODO : lien pour apprendre ?

Java Un autre langage très populaire, orienté objet et destiné à être compilé en bytecode exécuté sur une JVM (*Java Virtual Machine*). Très populaire dans les années 1995-2010. Pas enseigné en classes préparatoires, mais dans certaines écoles d'ingénieurs et des Universités. Android et la plupart de ses applications, mais aussi Minecraft sont programmées en Java. Ces ressources peuvent vous aider à apprendre : WikiBooks Java et les références données dedans.

C++ Le grand frère du langage C, qui est vraiment différent et n'est pas juste une extension. Très populaire depuis 1990, et un des langages les plus populaires même encore en 2021. Pas enseigné en classes préparatoires, mais dans certaines écoles d'ingénieurs et des Universités. La plupart des « gros » jeux vidéos sont programmés en C++, que ce soit pour consoles ou ordinateurs, notamment le Unreal Engine et Unity.

PHP Un autre langage pour programmer très populaire dans les années 2000-2015, surtout utilisé pour programmer des serveurs et des applications web dynamiques. Facebook et MediaWiki (qui fait fonctionner Wikipédia) sont notamment programmés en PHP.

Maple Un logiciel (propriétaire et payant) de calcul formel et un langage de programmation. Avant d'être remplacé par Python par la réforme des prépas scientifiques de 2013, il était enseigné en MPSI/MP, PCSI/PC et PSI. Très puissant pour faire des maths formelles, mais complètement inapproprié pour tout le reste.

MATLAB Un logiciel (propriétaire et payant) de calcul numérique vectoriel, et un langage de programmation. Avant d'être remplacé par Python par la réforme des prépas scientifiques de 2013, il était parfois utilisé en SI. Très puissant pour faire des maths numériques, mais complètement inapproprié pour tout le reste. Malgré son prix et ses défauts, il reste un standard dans beaucoup de domaines de l'ingénierie numérique et du traitement de signal, vous le croiserez sûrement dans la suite de vos études.

Julia Un logiciel libre et gratuit de calcul numérique vectoriel, et un langage de programmation moderne et puissant. Encore jeune mais déjà très populaire, je pense qu'il remplacera MATLAB dans quelques années. Si vous ne le croisez pas dans la suite de vos études, posez la question à tous vos profs qui vous forceraient à faire du MATLAB : *et pourquoi pas Julia ?*

13.3.3 Langages de production de documents

HTML Pour des pages web, utilisés avec JavaScript et CSS. Cf https://fr.wikipedia.org/wiki/HTML

CSS Pour la mise en page de pages web, utilisés avec HTML et JavaScript. Cf https://fr.wikipedia.org/wiki/Cascading%20Style%20Sheets

Markdown Un petit langage à balise, plus léger que HTML et LaTeX, qui permet d'écrire des documents modestes, ou des livres entiers (comme celui ci). Cf https://fr.wikipedia.org/wiki/Markdown

org-mode Le grand frère historique de Markdown, j'aime moins car moins standard désormais, et un peu trop « Emacs années 90 » pour moi. Il y a d'autres variantes à Markdown, notamment MediaWiki, AsciiDoc ou reStructuredText (rST).

TeX La base du logiciel LaTeX. Cf https://fr.wikipedia.org/wiki/TeX

LaTeX Le standard de rédaction de documents scientifiques destinés à être imprimés. En classes préparatoires, vous apprendrez les bases de l'écriture de formules mathématiques avec LaTeX, mais pas le reste, par exemple : $\forall x \in \mathbb{R}, \cos^2(x) + \sin^2(x) = 1$. Il existe plein de ressources pour apprendre, mais je recommande le WikiBooks LaTeX, et notamment la section sur les formules mathématiques.

LibreOffice Une suite de production de documents, contenant LibreOffice Writer, LibreOffice Impress pour les présentations, LibreOffice Draw pour le dessin, et LibreOffice Calc pour les feuilles de calcul (à la Excel). Cf https://fr.libreoffice.org/

Microsoft Word Pas un langage mais un outil, qui peut être téléchargé gratuitement sur différentes plateformes, ou utilisés en ligne. Néanmoins, il n'est pas libre. Je recommande plutôt l'alternative libre qui est LibreOffice. https://www.microsoft.com/fr-fr/microsoft-365/word

13.4 Formats de fichiers courants

PDF Pas un langage ou un outil de production de document, mais le format standard de documents qui sont terminés et destinés à être imprimés ou lus, avec la même apparence sur toutes les plateformes et tous les environnements. Si vous envoyez un devoir sous forme électronique, soit c'est du code, soit c'est un PDF : n'envoyez pas 7 pages numérisées comme 7 pièces jointes au format PNG ou JPEG. Un format produit par Adobe (qui produit notamment aussi Photoshop) non libre mais à la spécification publiée librement. La variante PDF/A est conçue pour un archivage à longue durée. https://fr.wikipedia.org/wiki/PDF

PNG Format d'images compressées sans pertes, principalement utilisées pour des captures d'écran et pour des logos. https://fr.wikipedia.org/wiki/JPEG

JPEG Format d'images compressées avec pertes, principalement utilisées pour les photographies. https://fr.wikipedia.org/wiki/JPEG

AVI Un format de stockage de vidéo avec du son, compressées avec pertes. TODO : ? https://fr.wikipedia.org/wiki/Audio_Video_Interleave

MKV Un format de stockage de vidéo avec du son, compressées avec pertes. TODO : ? https://fr.wikipedia.org/wiki/MPEG-1/2_Audio_Layer_3

MP3 Le format standard pour la musique compressée avec pertes. Format et logiciels non libres mais gratuits. TODO : ? https://fr.wikipedia.org/wiki/MPEG-1/2_Audio_Layer_3

Ogg Vorbis Un autre format de musique compressé avec pertes. Format libre, mais moins standard. TODO : ? https://fr.wikipedia.org/wiki/Vorbis

GIF Un format de stockage de petite vidéo sans son, compressées avec beaucoup de pertes. TODO : ? (prononcé « guiffe », [gif] en API). https://fr.wikipedia.org/wiki/Graphics_Interchange_Format

JSON Un format texte qui ressemble à un dictionnaire Python, stockage de clés/valeurs. C'est notamment le format utilisé par les notebooks Jupyter. Quasiment le standard pour l'échange de format texte par des programmes sur Internet. Cf https://json.org/.

13.5 Outils informatiques

Cf ma liste d'outils préférés https://perso.crans.org/besson/mes-outils-preferes.fr.html.

13.5.1 Site web

DuckDuckGo Moteur de recherche libre et non intrusif, qui ne vous espionne pas. Je vous recommande d'oublier Google et Bing.

Wikipédia L'encyclopédie libre en ligne, à laquelle tout le monde peut contribuer : https://fr.wikipedia.org/ !

SCEI Pour vos concours.

GitHub TODO :

YouTube TODO :

13.5.2 Outils génériques

Firefox Le navigateur web suggéré (cf https://www.mozilla.org/firefox/) pour lire ce livre en ligne et utiliser les notebooks Jupyter. Un des principaux navigateur utilisés au monde, et le dernier « grand » navigateur qui soit libre et indépendant, quasiment disponible sur toutes les plateformes, gratuitement. On peut lui ajouter plein d'extensions intéressantes, cf https://perso.crans.org/besson/firefox-extensions.fr.html pour mes suggestions.

Visual Studio Code L'éditeur de texte générique suggéré (cf https://code.visualstudio.com/) pour travailler en *C*, *OCaml* et *Python*. Un petit nouveau en comparaison des vénérables Emacs et Vi(m), mais bien plus moderne et à la prise en main plus facile. Disponible sur toutes les plateformes d'ordinateurs bureau et portable, gratuitement. On peut lui ajouter plein d'extensions intéressantes, cf https://perso.crans.org/besson/visualstudiocode.fr.html pour mes suggestions.

Jupyter notebook L'environnement recommandé pour programmer en *Python*, cf https://www.jupyter.org/. On peut s'en servir pour programmer en *OCaml* et en *C* TODO : liens.

Git Gestionnaire de version que vous prendrez en main en deuxième année. cf https://perso.crans.org/besson/tutogit.fr.html pour mes suggestions.

13.5.3 Compilateurs

ocamlc Compilateur de code *OCaml* vers du bytecode. A éviter, soit vous interprétez le code avec `ocaml` directement, soit vous compilez en code natif avec `ocamlopt`.

ocamlopt Compilateur de code *OCaml* vers du code natif. A préférer face à `ocamlc`.

ocamlc -i Pour afficher (ou générer) les signatures d'un ou plusieurs fichiers *OCaml*. TODO : donner ce genre de détails ici ? Ou ailleurs ?

GCC Le vénérable GNU *C* Compiler, pour compiler vos programmes en *C*. Nous utiliserons plutôt clang, qui est plus moderne et aux messages d'erreurs souvent moins cryptiques.

Clang Un autre compilateur *C*, pour compiler vos programmes en *C*. Nous le préférons à GCC, qui est plus ancien et aux messages d'erreurs souvent plus cryptiques.

13.6 Systèmes d'exploitations

Android Le principal système d'exploitation pour téléphone mobile en 2021, produit principalement par Google. On peut lui ajouter plein d'extensions intéressantes, cf https://perso.crans.org/besson/apk.fr.html pour mes suggestions.

iOS Un autre système d'exploitation pour téléphone mobile en 2021, produit par Apple. Apple vous espionne quasiment sur tous les aspects possibles, lorsque vous utilisez un appareil Apple, sachez le ! Je n'y connais rien, je ne pourrais pas vous aider.

Windows phone Un système d'exploitation pour téléphone mobile, quasiment disparu en 2021, produit par Microsoft. Microsoft vous espionne quasiment sur tous les aspects possibles, lorsque vous utilisez un appareil Microsoft, sachez le ! Je n'y connais rien, je ne pourrais pas vous aider.

Ubuntu Mon système d'exploitation pour ordinateur, bureau et portable, en 2021, produit par Canonical Ltd. Je vous recommande d'installer Ubuntu sur votre ordinateur, cf ce guide https://doc.ubuntu-fr.org/. C'est un système d'exploitation basée sur GNU/Linux.

Mac OS Un autre système d'exploitation pour ordinateur, bureau et portable, en 2021, produit par Apple. Apple vous espionne quasiment sur tous les aspects possibles, lorsque vous utilisez un appareil Apple, sachez le ! Je ne connais pas très bien, je ne pourrais pas vous aider aussi bien que sous Ubuntu.

Windows Un autre système d'exploitation pour ordinateur, bureau et portable, en 2021, produit par Microsoft. Le plus courant dans le grand public. Microsoft vous espionne quasiment sur tous les aspects possibles, lorsque vous utilisez un appareil Microsoft, sachez le ! Je connais pas trop mal, mais je ne pourrais pas vous aider aussi bien que sous Ubuntu.

13.7 Licences

Cf https://choosealicense.com/ pour plus d'informations.

MIT La licence de ce livre, cf https://lbesson.mit-license.org/.

Creative Commons Une autre licence libre très courante, cf https://creativecommons.org/licenses/by-nc-sa/4.0/deed.fr.

GPL Une autre licence libre très courante, celle qui régit les projets GNU, mais qui n'est pas très permissive.

CHAPITRE 14

Modèle de livre avec Jupyter Book

Un modèle de livre créé avec Jupyter Book, en français et avec des exemples complets d'utilisations des extensions Sphinx, Jupyter et Jupyter Book les plus utiles.

Avertissement : Work in progress...

==> Voir https://perso.crans.org/besson/Info-Prepas-MP2I/Modele-de-livre-avec-Jupyter-Book/ pour la version actuelle de ce livre.

14.1 Usage

14.1.1 Building the book

If you'd like to develop on and build the Modèle de livre avec Jupyter Book book, you should :
— Clone this repository and run
— Note : if you have GNU Make installed, just do

```
$ make setupvenv
$ make html
$ make preview
```

— Run `pip install -r requirements.txt` (it is recommended you do this within a virtual environment)
— (Recommended) Remove the existing `./_build/` directory
— Run `jupyter-book build ./`

A fully-rendered HTML version of the book will be built in `./_build/html/`.

14.1.2 Hosting the book locally

WARNING : I **don't want** to use any GitHub actions workflow : I want to compile my books locally, on my machine, and host them (without version control) on my websites.

The book is hosted at https ://perso.crans.org/besson/Info-Prepas-MP2I/Modele-de-livre-avec-Jupyter-Book/

> Why ? Using automated build is not ecological friendly, I want to be in full control of the builds. And using GitHub pages for a book that changes very regularly is also unefficient. I will probably add this automated compilation of the books, when they will be mature enough, and when only the content will change, not the configurations.

14.1.3 Hosting the book on GitHub pages

The html version of the book is **NOT** hosted on the `gh-pages` branch of this repo. A GitHub actions workflow has been created that automatically builds and pushes the book to this branch on a push or pull request to main.

If you wish to disable this automation, you may remove the GitHub actions workflow and build the book manually by :
— Navigating to your local build ; and running,
— `ghp-import -n -p -f Modèle de livre avec Jupyter Book/_build/html`
This will automatically push your build to the `gh-pages` branch. More information on this hosting process can be found here.

14.2 Contributors

We welcome and recognize all contributions. You can see a list of current contributors in the contributors tab.

14.3 Credits

This project is created using the excellent open source Jupyter Book project and the executablebooks/cookiecutter-jupyter-book template.

14.4 :scroll : License ?

CHAPITRE 15

Choses à faire (TODO list)

15.1 First week

— [x] write first version of requirements.txt, and Makefile
— [x] install Jupyter-book and extensions in virtualenv
— [ ] write complete versions of requirements.txt, and Makefile
— [x] start fresh jupyter-book with command

```
jupyter-book create --cookiecutter
```

— [x] build the site, in HTML singlepage (not dirhtml)
— [ ] try each builder :
    — [x] make dirhtml : okay, but I don't see the point so won't use
    — [x] make pdfhtml DONE : it works, but it's not so pretty. It can be useful but not great, without cheating with this config
    — [ ] make latex
    — [ ] make pdflatex
    — [ ] make linkcheck
— [ ] write demo pages
— [ ] for each extensions
    — [ ] install it, try it, add it to requirements.txt
    — [ ] write an example on the demo page
— [ ] License Creative Commons instead of MIT ?
— [ ] Take inspirations from this book in French
    — https ://github.com/miti-gt-donnees/guide/blob/master/_config.yml
— [ ] create my own logo.png ?
— [ ] for notebooks, never include Table of Contents cell from jupyter extension : it displays wrongly in the built book
— [ ] for notebooks, how come links are not linked by default ? (https ://perso.crans.org/besson/Info-Prepas-MP2I/Modele-de-livre-avec-Jupyter-Book/notebooks/Exemple%20de%20notebook%20avec%20Python.html#pour-en-apprendre-plus)

15.2 How to cleanly add TODO ? - FAILED

Using a reST directive ? No it doesn't work :

Using a reST directive and a `eval-rst` directive from Markdown (MyST) :

Using a MyST directive directly ? No it broke the `pdfhtml` builder.

And printing the list of TODO :

See https://www.sphinx-doc.org/en/master/usage/extensions/todo.html

15.3 Demo pages

These pages should render perfectly, otherwise something is broken in the jupyter-book setup.
- TODO : how to write clean intern links
- [x] See *MyST cheat sheet* ;
- [x] See *1 reStructuredText Demonstration* ;
- [x] DONE write demo for other kernels : Python 3, C, OCaml, Java, Shell/Bash, SQL :
    - [x] notebooks/index ;
    - [x] are they all correctly included ?
    - [ ] See SQL ;
    - [ ] Fix example of C ;

CHAPITRE 16

Utterances for interactive comments on the webbook

Utterances is a commenting engine built on top of GitHub Issues. It embeds a comment box in your page that users (with a GitHub account) can use to ask questions. These become comments in a GitHub issue in a repository of your choice.

Note : Utterances should be activated on this page. You can see the comment box at the bottom of this page's content. Click the « log in » button and you'll be able to post comments !

16.1 Activate `utterances`

You can activate `utterances` by adding the following to your `_conf.yml` file :

```
html:
  comments:
    utterances:
      repo: "github-org/github-repo"
```

Note that the `utterances` UI will not show up when you are previewing your book locally, it must be hosted somewhere on the web to function.

16.2 Configure `utterances`

You can pass optional extra configuration for utterances. You may do so by providing `key:val` pairs alongside `repo:` in the configuration. See the utterances documentation for your options.

When you build your documentation, pages will now have a comment box at the bottom. If readers log in via GitHub they will be able to post comments that will then map onto issues in your GitHub repository.

16.3 Activating utterances only on this page

This HTML activates utterances only on this page

````
```{raw} html
<script
   type="text/javascript"
   src="https://utteranc.es/client.js"
   async="async"
   repo="Info-Prepas-MP2I/Modele-de-livre-avec-Jupyter-Book"
   issue-term="pathname"
   theme="github-light"
   label=" comment"
   crossorigin="anonymous"
/>
```
````

Bibliographie

[PGH11] Fernando Perez, Brian E Granger, and John D Hunter. Python : an ecosystem for scientific computing. *Computing in Science & Engineering*, 13(2) :13–21, 2011.

[CIT2002] Citations are text-labeled footnotes. They may be rendered separately and differently from footnotes.

[PGH11] Fernando Perez, Brian E Granger, and John D Hunter. Python : an ecosystem for scientific computing. *Computing in Science & Engineering*, 13(2) :13–21, 2011.

Index

M

O

P

R

S

T

U

V

W

Y