
Jupman

A template manager for online books made with Jupyter notebooks and NBSphinx doc generator

People That Write a Lot

Oct 16, 2020

Copyright © 2020 by People That Write a Lot.

Jupman is available under the Creative Commons Attribution 4.0 International License, granting you the right to copy, redistribute, modify, and sell it, so long as you attribute the original to People That Write a Lot and identify any changes that you have made. Full terms of the license are available at:

<http://creativecommons.org/licenses/by/4.0/>

The complete book can be found online for free at:

<https://jupman.softpython.org/en/themed/>

CONTENTS

About	1
Preface	2
Revisions	2
1 Overview	3
1.1 Contents	3
1.2 Credits	3
2 Jupman Usage	5
2.1 Installation	5
2.2 Getting Started	7
2.3 Building the manual	8
2.4 Publishing	8
2.5 Editing the worksheets	9
2.6 Website	14
2.7 Exams	16
2.8 Developer notes	18
3 Jupman Tests	19
3.1 Sezione 1	19
3.2 Sezione 2	19
3.3 Quotes	19
3.4 Download links	20
3.5 Info/Warning Boxes	20
3.6 Math	20
3.7 Unicode	21
3.8 Image	21
3.9 Expressions list	22
3.10 Toggable cells	23
3.11 HTML details in Markdown, code tag	23
3.12 Files in templates	24
3.13 Python tutor	24
3.14 Stripping answers	27
3.15 Metadata to HTML classes	27
3.16 Formatting problems	27
4	33
5 Chapter examples	35
5.1 Python example	35
5.2 Jupyter example	37

5.3	Jupyter and python example	41
6	Templates	45
6.1	Changelog	45
6.2	Past Exams	47
6.3	Exam project	47
6.4	Project ideas	48
6.5	Jupman Project	49
6.6	Markdown	50
7	Index	55

About

A template manager for online books made with [Jupyter](#)¹ notebooks and [NBSphinx](#)² doc generator.

Features:

- Based on [NBSphinx](#)³ which produces website made of static files
- Supports build with ReadTheDocs or [Github Actions](#)⁴ (or local Docker emulating ReadTheDocs)
- builds exercises from solution templates (both .ipynb and .py)
- builds chapter zips
- supports sharing code among chapters
- Python Tutor integration (can work offline, doesn't need to install dependencies)
- includes a exam management system (script and grades spreadsheet)
- configuration clearly separated from code
- decent PDF layout
- made for Python 3
- comes with *documentation*
- Open source code on [Github](#)⁵
- Apache License v2.0

Currently lacking:

- Python Tutor doesn't work in JupyterLab
- more testing, especially for EPUB support and exam management

Used by:

- [SoftPython book \(english\)](#)⁶
- [SoftPython book \(italian\)](#)⁷
- [Scientific Programming Lab at University of Trento, Data Science Master](#)⁸ (English)

¹ <http://jupyter.org>

² <http://nbsphinx.readthedocs.io/>

³ <http://nbsphinx.readthedocs.io/>

⁴ <https://github.com/DavidLeoni/readthedocs-to-actions>

⁵ <https://github.com/DavidLeoni/jupman>

⁶ <https://en.softpython.org/>

⁷ <https://it.softpython.org/>

⁸ <https://sciprogram.davidleoni.it/>

Preface

This book is the result of ... We thank this and that ...

Revisions

- **16 October 2020:** Released v3.2
- **16 January 2020:** Released v3.1
- **29 December 2019:** Released v3.0
- **24 September 2018:** Released v2.0
- **3 August 2018:** Released v0.8
- *Change log*

OVERVIEW

1.1 Contents

1. *JUPMAN USAGE*
2. Chapter examples
 1. *Python example*
 2. *Jupyter example*
 3. *Jupyter and python example*
3. Templates
 1. *Past exams*
 2. *Changelog*

1.2 Credits

- This site was made with Jupyter using NBSphinx extension⁹ and Jupman template¹⁰.

⁹ <http://nbsphinx.readthedocs.io/>

¹⁰ <http://jupman.readthedocs.io/>

JUPMAN USAGE

Jupyter Python 3 worksheets build system and exam manager. See Jupman manual at jupman.readthedocs.io¹¹

Jupman uses [NbSphinx](http://nbsphinx.readthedocs.io/)¹² and either [ReadTheDocs](https://readthedocs.org)¹³ or [Github Actions](https://github.com/features/actions)¹⁴

2.1 Installation

(Instructions are for Ubuntu, on Windows may differ)

First, on Github, fork as a template [jupman project](https://github.com/DavidLeoni/jupman)¹⁵ to create yours, for example `my-project`.

Then, on your computer, clone the `my-project` from Github

You can choose to build either on:

- [ReadTheDocs](https://readthedocs.org)
- [Github Actions](https://github.com/features/actions)
- locally with plain Sphinx
- locally with [RTD Docker](https://github.com/DavidLeoni/readthedocs-to-actions)¹⁶

(Note Jupman itself is building on both [ReadTheDocs](https://readthedocs.org) and [Github Actions](https://github.com/features/actions) only for testing purposes, one is enough)

2.1.1 To build with ReadTheDocs:

IMPORTANT: choose a name which is NOT already on [ReadTheDocs](https://readthedocs.org)¹⁷

Create a [ReadTheDocs account](https://readthedocs.org)¹⁸ using the same name as in Github so the address in readthedocs will be something like `my-project.readthedocs.org`.

- Use [ReadTheDocs](https://readthedocs.org) panels to link the project to your Github repository.
- In *Admin->Advanced settings panel*, set:
 - *Python interpreter* to `CPython 3.x`
 - *Requirements* to `requirements-build.txt`

¹¹ <http://jupman.readthedocs.io>

¹² <http://nbsphinx.readthedocs.io/>

¹³ <https://readthedocs.org>

¹⁴ <https://github.com/features/actions>

¹⁵ <https://github.com/DavidLeoni/jupman>

¹⁶ <https://github.com/DavidLeoni/readthedocs-to-actions>

¹⁷ <http://readthedocs.org>

¹⁸ <http://readthedocs.org>

2.1.2 To build with Github Actions:

Configure `.github/workflows/main.yml`¹⁹ on your computer to your needs - you will need to:

1. at the beginning there is an `if` which makes the workflow only work in DavidLeoni/jupman repository, change it with your names
2. set `RTD_PRJ_NAME`
3. If you want to publish to [Github Pages](https://pages.github.com/)²⁰: everything is set, just create an empty branch `gh-pages` in a new `HTML_FOLDER` **before** committing - from some other folder in your file system:

```
git clone YOUR_REPO_ADDRESS HTML_FOLDER
cd HTML_FOLDER
git checkout --orphan gh-pages
git rm -rf .
touch bla
git add .
git commit -m "init"
git push origin gh-pages
```

2.1.3 Local build with Sphinx

1. Install Python 3.5+
2. [Install Jupyter](http://jupyter.org/install.html)²¹
3. Install Python modules -from the root of the project, run:

```
python3 -m pip install --user -r requirements-build.txt
```

This will install required modules in your home directory

2.1.4 Optional - Running tests

To check everything is working, you may want to run the tests.

1. Install test dependencies:

```
python3 -m pip install --user -r _test/requirements-test.txt
```

2. Run the tests:

```
python3 -m pytest _test/*_test.py
```

¹⁹ <https://github.com/DavidLeoni/jupman/blob/master/.github/workflows/main.yml>

²⁰ <https://pages.github.com/>

²¹ <http://jupyter.org/install.html>

2.1.5 Optional - Install Jupyter contrib extensions

For a better editing experience like having Table of contents and other things, do the following:

1. install the [Jupyter contrib extensions](#)²² package:

If you have Anaconda:

```
conda install -c conda-forge jupyter_contrib_nbextensions
```

If you don't have Anaconda:

```
python3 -m pip install --user jupyter_contrib_nbextensions
```

2. Install it in Jupyter:

```
jupyter contrib nbextension install --user
```

3. Enable extensions (putting here recommended ones):

For being able to view table of contents while editing notebooks, install `toc2` extension:

```
jupyter nbextension enable toc2/main
```

To see tocs when in a document you will need to press a list button at the right-end of the toolbar).

(since Jupman 0.8 custom injected tocs are disabled by default)

4. For a nice GUI to install extensions, install the [Jupyter Nbextensions configurator](#)²³:

If you have Anaconda:

From Anaconda Prompt:

```
conda install -c conda-forge jupyter_nbextensions_configurator
```

If you don't have Anaconda:

```
python3 -m pip install --user jupyter_nbextensions_configurator
```

After installing, enable it:

```
jupyter nbextensions_configurator enable --user
```

and then start Jupyter, in file browser look for a Nbextensions tab

2.2 Getting Started

1. Edit as needed `conf.py`²⁴, which is the configuration file for Sphinx. In particular, you **MUST** edit the sections marked with `TODO`
2. Try to launch a build

```
python3 build.py
```

²² https://github.com/ipython-contrib/jupyter_contrib_nbextensions

²³ https://github.com/Jupyter-contrib/jupyter_nbextensions_configurator

²⁴ <https://github.com/DavidLeoni/jupman/blob/master/conf.py>

For more info, see *related section*

3. If everything works fine on your computer, push changes back to Github
4. Go back to ReadTheDocs and try to run a build. Hopefully your project will become available on something like *my-project.readthedocs.org*
5. If you want to grade exams, see *Exams* section.

You should now be ready to create your notebooks by launching from the project root:

```
jupyter notebook
```

6. If you wish your notebooks to appear in the generated manual, you have to add them in the `toc.rst` file.

NOTE: the page `toc-page.rst`²⁵, which is set to be the `master_doc` of Sphinx, will just load the actual Table of Contents which is in `toc.rst`²⁶. It looks a bit convoluted because when it comes to indexes Sphinx is not much reliable, see [this issue](#)²⁷. We strongly advise *not* to change these settings !

7. edit the home, which is in the `index.ipynb`²⁸ file

2.3 Building the manual

For quick build that only produces html:

```
python3 build.py -q
```

Site will be created in `_build/` folder.

For help:

```
python3 build.py -h
```

To build everything (html + pdf + epub), go to console and from the root of the directory run:

```
python3 build.py
```

NOTE: to also generate PDF you will need to install Latex environment

2.4 Publishing

For publishing, the system uses ReadTheDocs so it is enough to push to master and ReadTheDocs will do the rest (for example, for `jupman` it is at address jupman.readthedocs.io²⁹)

IMPORTANT: ReadTheDocs WILL **NOT** execute Jupyter notebooks because of [this bug](#)³⁰

²⁵ <https://github.com/DavidLeoni/jupman/blob/master/toc-page.rst>

²⁶ <https://github.com/DavidLeoni/jupman/blob/master/toc.rst>

²⁷ <https://github.com/DavidLeoni/jupman/issues/11>

²⁸ <https://github.com/DavidLeoni/jupman/blob/master/index.ipynb>

²⁹ <http://jupman.readthedocs.io>

³⁰ <https://github.com/DavidLeoni/softpython/issues/2>

2.5 Editing the worksheets

Here we give an overview of how to edit worksheets. More info can be found in *Jupman tests notebook*

2.5.1 Common files

There are a bunch of files common to all worksheets and possibly website

You do not need to change them (except maybe my_lib.py)

File	Description	Jupyter	Website
jupman.py ³¹	utilities for worksheets	X	
my_lib.py ³²	custom utilities for worksheets (you can change the name)	X	
_static/js/jupman.js ³³	Javascript code	X	X
_static/css/jupman.css ³⁴	CSS	X	
_static/css/jupman-web.css ³⁵	CSS		X

2.5.2 Running Jupyter

First of all, run Jupyter from the root of the directory:

```
jupyter notebook
```

2.5.3 Source code for chapters

Put chapters one per folder, in the root. Any folder which does not starts with underscore _ or exam/ will be considered a chapter.

During build, each chapter gets automatically zipped and zip goes to _static/generated. So for example, python-example/ produces a zip called _static/generated/python-example.zip, which will have these contents:

```
python-example
  _static
    js
      jupman.js
      toc.js
    css
      jupman.css
    img
      cc-by.png
  python-example.ipynb
  lab.py
  lab_test.py
  lab_sol.py
  jupman.py
  my_lib.py
```

³¹ <https://github.com/DavidLeoni/jupman/blob/master/jupman.py>

³² <https://github.com/DavidLeoni/jupman/blob/master/jupman.py>

³³ https://github.com/DavidLeoni/jupman/blob/master/_static/js/jupman.js

³⁴ https://github.com/DavidLeoni/jupman/blob/master/_static/css/jupman.css

³⁵ https://github.com/DavidLeoni/jupman/blob/master/_static/css/jupman-web.css

The zip folder structure will be a merge of chapter files and files shared by all chapters which are specified in `exercises_common_files` variable in `conf.py`. Since the root in the zip becomes the chapter itself, `jupman` will process `.py` and `.ipynb` files for fixing eventual relative imports. Markdown and HTML links in `ipynb` will also be adjusted.

Exercise files can be automatically generated from solutions, as we will see next.

2.5.4 Type of exercises

There can be three kinds of exercises: exercises in python files, exercises in jupyter files and mixed jupyter and python exercises.

You can automatically generate an exercise from a solution file by stripping text marked with special tags. You can inspect generated files in `_build/jupman/` directory

On the website, students will be able to see solutions by clicking on appropriate buttons.

In the zips to download, two versions of files will be provided, one without solution and one with solutions (in exam modality of course no solution will be shipped)

Exercises in python files

See *`python-example/python-example.ipynb`*

In this type of exercises, typically you have a Jupyter file (like `python-example.ipynb`) that describes the exercise and then the actual exercises are in python files.

If there is a solution file `FILE_sol.py` ending in `_sol.py` but no corresponding exercise file `FILE.py` without the `_sol`:

then `Jupman` will try to generate `FILE.py` one from `FILE_sol.py`. To do so, it will look for tags to strip inside the solution file.

If there is already an exercise file like this:

- `python_intro.py`
- `python_intro_sol.py`

`Jupman` will just copy the existing file.

Exercises in Jupyter files

See example: *`jupyter-example/jupyter-example-sol.ipynb`*

This type of exercises stay in a jupyter notebook itself.

If there is a notebook ending in `-sol.ipynb`, the following applies (**WARNING:** for `ipynb` files we use dash `-`, *not* the underscore `_`):

- the notebook must contain tags to strip
- exercises derived will have 'EXERCISES' appended to the title (the word can be customized in `conf.py` - you might need to translate it)

Mixed exercises in Jupyter and Python files

See *jup-and-py-example/jup-and-py-example-sol.ipynb*

2.5.5 Tags to strip

Start tags begin with a # while end tags begin with a #\

jupman-raise

Replaces code inside with an Exception (text is customizable in `conf.py`). Be careful to position the comment exactly with the indentation you want the raise to appear. For example:

```
def add(x, y):
    #jupman-raise
    return x + y
    #/jupman-raise
```

becomes

```
def add(x, y):
    raise Exception('TODO IMPLEMENT ME !')
```

jupman-strip

Just strips code inside

```
def f(x):
    print(x)

#jupman-strip
def help_func(x, y):
    return x - y
#/jupman-strip

def g(y):
    return y
```

becomes

```
def f(x):
    print(x)

def g(y):
    return y
```

write here

This special tag for python code erases whatever is found afterwards the `# write here` comment

- you can put how many spaces you want in the comment
- phrase can be customized in `conf.py`

```
w = 5

# write here

x = 5 + w
y = 2 + x
```

becomes

```
w = 5

# write here
```

SOLUTION

In a code cell, if you put `# SOLUTION` at the beginning the whole cell content gets deleted (`# SOLUTION` string included).

- Word can be customized in `conf.py`

```
# SOLUTION

def f():
    print('hello')
```

becomes nothing:

```
..
```

QUESTION - ANSWER

In a markdown cell, everything in ‘**ANSWER:**’ cell will be stripped.

- Markdown can be customized in `conf.py`

QUESTION: Describe why iPhone 8 is better than 7

```
<div id="jupman-sol-25" class="jupman-sol jupman-sol-question" onclick="jupman.toggleSolution('jupman-sol-25');">
  <input type="button" class="jupman-sol-label" data-jupman-show="Show answer" data-jupman-hide="Hide" value="Show answer"/>
  <div class="jupman-sol-content" style="display:none">
```

ANSWER: it costs more

```
</div> </div>
```

becomes:

QUESTION: Describe why iPhone 8 is better than 7

2.5.6 Utilities and custom js and css

If you need custom js and/or css in a notebook, you can inject it by running `jupman.init()` in the first cell

NOTE: it is not really mandatory, it's mostly intended to tweak notebooks downloaded locally. It should be avoided for notebooks meant for students, as it is more likely it will mess their configurations - also, they might copy the notebooks without knowing they contain the custom js and use weird extensions which could generate conflicts (such as double toc)

For notebooks in the root folder:

```
import jupman
jupman.init()
```

Worksheets in subfolders can use `sys.path` to locate the module

```
import sys
sys.path.append('../')
import jupman
jupman.init()
```

If you think it looks ugly, see [this issue](#)³⁶ for why we don't use alternatives such as modules and relative imports.

Show table of contents: Since 0.8, toc is disabled. If you want it, try to *install toc2 extension*, otherwise you can still enable jupman toc with `jupman_init(toc=True)`. Running it will create the sidebar even when editing in Jupyter. If you want to refresh the sidebar, just run again the cell. It is not recommended, though, especially in notebooks meant to be shipped to students (see considerations above).

2.5.7 Hiding cells

To hide cells (like for example the `import jupman` code), click View->Cell toolbar -> Edit metadata and add "nbsphinx": "hidden" to the JSON (see also original [NBSphinx docs](#)³⁷ and *Toggable cells in Jupman tests*).

NOTE: As of NBSphinx 2.17, it is not possible to hide only cell text but not the output.

Implications of hiding 'import jupman'

Only in the HTML version, hiding the `import jupman` code, will also prevent `jupman.py` to embed inside the page the Javascript file `jupman.js`: this is perfectly fine as it is fetched separately thanks to the `app.add_javascript('js/jupman.js')` command in `conf.py`

2.5.8 Launch unit tests

Inside worksheets you can run `unittest` tests.

To run all the tests of a test class, write like this

```
jupman.run(NameOfTheTestClass)
```

To run a single method, write like this:

```
jupman.run(NameOfTheTestClass.nameOfTheMethod)
```

³⁶ <https://github.com/DavidLeoni/jupman/issues/12>

³⁷ <https://nbsphinx.readthedocs.io/en/0.2.14/hidden-cells.html#Hidden-Cells>

2.5.9 Python Tutor

Among the various ways to embed Python Tutor, we decided to implement a special `jupman.pytut()` method. First you need to import the `jupman` module:

```
[2]: import jupman
```

Then you can put a call to `jupman.pytut()` at the end of a cell, and the cell code will magically appear in python tutor in the output (except the call to `pytut()` of course). To see Python Tutor you don't need to be online

```
[3]: x = [5,8,4]
     y= {3:9}
     z = [x]

     jupman.pytut()
```

```
[3]: <IPython.core.display.HTML object>
```

Beware of variables which were initialized in previous cells, they won't be available in Python Tutor and you will get an error:

```
[4]: w = 8
```

```
[5]: x = w + 5
     jupman.pytut()
```

```
Traceback (most recent call last):
  File "/home/da/Da/prj/jupman/prj/jupman.py", line 2305, in _runscript
    self.run(script_str, user_globals, user_globals)
  File "/usr/lib/python3.5/bdb.py", line 431, in run
    exec(cmd, globals, locals)
  File "<string>", line 2, in <module>
NameError: name 'w' is not defined
```

```
[5]: <IPython.core.display.HTML object>
```

2.6 Website

2.6.1 customize theme

If you want to change site colors and other changes, edit `_static/css/jupman-theme.css`³⁸

³⁸ https://github.com/DavidLeoni/jupman/blob/master/_static/css/jupman-theme.css

2.6.2 Fonts

Fonts are a bit of a complex topic

TODO this part is just a collection of personal notes

- The missing guide to font formats³⁹
- <https://docs.readthedocs.io/en/latest/guides/adding-custom-css.html>
- RTD Code font issue on github⁴⁰

Tools:

Comprehensive article: <https://www.useragentman.com/blog/2011/02/20/converting-font-face-fonts-quickly-in-any-os/> and <https://www.useragentman.com/blog/the-css3-font-converter/>

<https://github.com/zoltan-dulac/css3FontConverter>

woff2

<https://github.com/google/woff2>

sfnt2woff

```
sudo apt-get install libbrotli-dev
sfnt2woff SomeFont.otf
```

mkeot

```
sudo apt-get install eot-utils
mkeot SomeFont.otf > SomeFont.eot
```

or <https://github.com/wget/ttf2eot>

FontForge (GUI and scriptable)

```
sudo apt-get install fontforge
```

2.6.3 font sizes

<https://www.24a11y.com/2019/pixels-vs-relative-units-in-css-why-its-still-a-big-deal/>

<https://chiamakaikanyi.dev/sizing-in-css-px-vs-em-vs-rem/>

2.6.4 Warning about old versions

ReadTheDocs has a mechanism⁴¹ to warn the user if he's looking at an old version of the site, but we found it doesn't work much for course-based documentation. So for versioning we think it's better to adopt a mixed git branch / tags development model, and we added a template warning to show in old branches. To enable it in an old branch, just rename `_templates/breadcrumbs.html.bak` into `_templates/breadcrumbs.html` and edit as needed.

³⁹ <https://creativemarket.com/blog/the-missing-guide-to-font-formats>

⁴⁰ https://github.com/readthedocs/sphinx_rtd_theme/issues/524

⁴¹ <https://docs.readthedocs.io/en/latest/versions.html>

2.7 Exams

Jupman comes with a script to manage exams called `exam.py`, which allows to manage the full cycle of an exam.

2.7.1 What is an exam

Exam text is represented as Jupyter notebooks, which are taken from `_templates/exam/solutions/exam-yyyy-mm-dd.ipynb`

Exercises for students: they are supposed to be the exam notebook itself and / or plain python files (or the notebook itself) plus unittests and relative solutions.

Marks spreadsheet: By default there is also an LibreOffice spreadsheet to give marks, in case you need it.

When you initialize an exam with the `init` command, for example for date `2000-12-31`, all the presets in `_templates/exam/` are copied to `private/2000-12-31/` and `private/2000-12-31/solutions`. Presets can be changed at will to suit your needs. When packaging, student zip is assembled in `private/2000-12-31/student-zip`

System is flexible enough so you can privately work on next exams in `private/` folder and still being able to publish modifications to main website. After an exam, you can copy the private exam to the public folders in `past-exams/`.

2.7.2 Exam commands

To see the help:

```
python3 exam.py -h
```

To see help for a particular subcommand, like i.e. `init`, type the subcommand followed by `-h`:

```
python3 exam.py init -h
```

Running commands should be quite self-explanatory.

NOTE: as of today (Dec 2019) software may contain bugs, but at least we check for major misuses (like trying to overwrite existing exams).

In the file `create-exam-example.sh` there is a typical run of the script, which creates the example exam for date `2000-12-31`. Notice it might ask you to delete the existing `2000-12-31` exam, if it does just follow the instructions. Here is the output:

```
$ ./create-exam-example.sh
python3 exam.py init 2000-12-31

You can now edit Python solutions, tests, exercises and exam notebook here :

    _private/2000-12-31/solutions

DONE.

python3 exam.py package 2000-12-31
Cleaning _private/2000-12-31/server/jupman ...
Copying built website ...
Building pdf ..
Copying exercises to _private/2000-12-31/student-zip/jupman-2000-12-31-FIRSTNAME-
↪LASTNAME-ID/
```

(continues on next page)

(continued from previous page)

```

Copying code
  from _private/2000-12-31/solutions
  to   _private/2000-12-31/student-zip/jupman-2000-12-31-FIRSTNAME-LASTNAME-ID/
Writing (patched) _private/2000-12-31/student-zip/jupman-2000-12-31-FIRSTNAME-
↪LASTNAME-ID/exam-2000-12-31.ipynb
Generating _private/2000-12-31/student-zip/jupman-2000-12-31-FIRSTNAME-LASTNAME-
↪ID/trees.py
Writing _private/2000-12-31/student-zip/jupman-2000-12-31-FIRSTNAME-LASTNAME-ID/
↪example.txt
Generating _private/2000-12-31/student-zip/jupman-2000-12-31-FIRSTNAME-LASTNAME-
↪ID/lists.py
Writing (patched) _private/2000-12-31/student-zip/jupman-2000-12-31-FIRSTNAME-
↪LASTNAME-ID/trees_test.py
Writing (patched) _private/2000-12-31/student-zip/jupman-2000-12-31-FIRSTNAME-
↪LASTNAME-ID/lists_test.py
Creating dir _private/2000-12-31/student-zip/jupman-2000-12-31-FIRSTNAME-LASTNAME-
↪ID/img
Writing _private/2000-12-31/student-zip/jupman-2000-12-31-FIRSTNAME-LASTNAME-ID/
↪img/mountains.jpg
Creating student exercises zip: _private/2000-12-31/server/jupman-2000-12-31-exam.
↪zip
Writing jupman.py
Writing my_lib.py
Writing _static/img/cc-by.png
Writing _static/js/jupman.js
Writing _static/css/jupman.css
Writing _static/js/toc.js
Wrote _private/2000-12-31/server/jupman-2000-12-31-exam
Creating server zip: _private/2000-12-31/jupman-2000-12-31-server.zip

You can now browse the website at:  /home/da/Da/prj/jupman/prj/_private/2000-12-31/
↪server/jupman/html/index.html

DONE.

```

```

----- Simulating some shipped exams...
mkdir -p _private/2000-12-31/shipped/john-doe-112233
cp _templates/exam/solutions/lists_sol.py _templates/exam/solutions/lists_test.py _
↪templates/exam/solutions/trees_sol.py _templates/exam/solutions/trees_test.py _
↪private/2000-12-31/shipped/john-doe-112233
mkdir -p _private/2000-12-31/shipped/jane-doe-445566
cp _templates/exam/solutions/lists_sol.py _templates/exam/solutions/lists_test.py _
↪templates/exam/solutions/trees_sol.py _templates/exam/solutions/trees_test.py _
↪private/2000-12-31/shipped/jane-doe-445566
----- Done with shipped exams simulation, time to grade ...

python3 exam.py grade 2000-12-31
  Copying Python files to execute and eventually grade in _private/2000-12-31/graded/
↪john-doe-112233/graded
  Copying original shipped files (don't touch them!) in _private/2000-12-31/graded/
↪john-doe-112233/shipped
  Copying Python files to execute and eventually grade in _private/2000-12-31/graded/
↪jane-doe-445566/graded
  Copying original shipped files (don't touch them!) in _private/2000-12-31/graded/
↪jane-doe-445566/shipped

```

(continues on next page)

(continued from previous page)

```

DONE.

python3 exam.py zip-grades 2000-12-31

You can now find zips to send to students in _private/2000-12-31/graded

DONE.

python3 exam.py publish 2000-12-31
Copying solutions to exams/2000-12-31/solutions
Copying exam PDF text

Exam Python files copied.

You can now manually build and run the following git instructions to publish the
↪exam.
    ./build.py
    git status # just to check everything is ok
    git add .
    git commit -m 'published 2000-12-31 exam'
    git push

DONE.

Finished example exam run !!

```

2.8 Developer notes

2.8.1 Fix nbsphinx to create rst files

Sometimes nbsphinx does not report properly RST conversion errors (see [bug⁴²](https://github.com/DavidLeoni/jupman/issues/9)). As a hacky workaround, you might take the `nbsphinx.py` from `~/local/lib/python3.5/site-packages/`, make a copy of it in your project home and patch it [like this⁴³](https://github.com/DavidLeoni/jupman/commit/0f332629ce4e2b0186c954c55aea7fa67992ace9#diff-bd3d9c4d2e80ed83fd2443d1301aa65bR649). When you call sphinx, it will generate RST files in `_build/jupman-rst/`.

Of course, things can be cleaner using a virtual env [with venv⁴⁴](https://docs.python.org/3/library/venv.html)

```
[ ]:
```

⁴² <https://github.com/DavidLeoni/jupman/issues/9>

⁴³ <https://github.com/DavidLeoni/jupman/commit/0f332629ce4e2b0186c954c55aea7fa67992ace9#diff-bd3d9c4d2e80ed83fd2443d1301aa65bR649>

⁴⁴ <https://docs.python.org/3/library/venv.html>

JUPMAN TESTS

In this page we put some tests for Jupyter. The page Title has one sharp, the Sections always have two sharps.

3.1 Sezione 1

bla bla

3.2 Sezione 2

Subsections always have three sharps

3.2.1 Subsection 1

bla bla

3.2.2 Subsection 2

bla bla

3.3 Quotes

I'm quoted with **greater than** symbol on multiple lines Am I readable?

```
I'm quoted with **spaces**  
on multiple lines  
Am I readable?
```

3.4 Download links

Files manually put in `_static`:

- Download [trial.odt](#)
- Download [trial.pdf](#)

Files in arbitrary folder position :

- Download [requirements.txt](#)

NOTE: download links are messy, [see issue 8](#)⁴⁵

3.5 Info/Warning Boxes

Until there is an info/warning extension for Markdown/CommonMark (see this issue), such boxes can be created by using HTML

elements like this:

Note: This is an info!

Note: This is a warn!

For this to work reliably, you should obey the following guidelines:

- The class attribute has to be either “alert alert-info” or “alert alert-warning”, other values will not be converted correctly.
- No further attributes are allowed.
- For compatibility with CommonMark, you should add an empty line between the start tag and the beginning of the content.

3.6 Math

For math stuff, [see npshpinx docs](#)⁴⁶

Here we put just some equation to show it behaves fine in Jupman

This is infinity: ∞

⁴⁵ <https://github.com/DavidLeoni/jupman/issues/8>

⁴⁶ <https://nbsphinx.readthedocs.io/en/0.2.14/markdown-cells.html#Equations>

3.7 Unicode

Unicode characters should display an HTML, but with latex you might have problems, and need to manually map characters in conf.py

You should see a star in a black circle: ☉

You should see a check: ✓

table characters: | |└┐

3.8 Image

3.8.1 SVG Images

SVG images work in notebook, but here it is commented since it breaks Latex, [see issue⁴⁷](#)

```
! [An image] (img/cc-by.svg)
```

This one also doesn't work (and shows ugly code in the notebook anyway)

```
from IPython.display import SVG
SVG(filename='img/cc-by.svg')
```

3.8.2 PNG Images



3.8.3 Inline images - pure markdown

```
Bla ![A PNG image] (_static/img/notebook_icon.png) bli blo
```



Bla

bli blo

⁴⁷ <https://github.com/DavidLeoni/jupman/issues/1>

3.8.4 Inline images - markdown and img

```
bla  bli blo
```



3.8.5 Img class

If we pass a class, it will to be present in the website:

```

```



3.9 Expressions list

Highlighting **does** work both in Jupyter and Sphinx

Three quotes, multiple lines - Careful: put **exactly 4 spaces** indentation

```
1. [2, 3, 1] != "[2, 3, 1]"
```

```
2. [4, 8, 12] == [2*2, "4*2", 6*2]
```

```
3. [][:] == []
```

Three quotes, multiple lines, more compact - works in Jupyter, **doesn't** in Sphinx

```
1. python [2, 3, 1] != "[2, 3, 1]"
```

```
2. python [4, 8, 12] == [2*2, "4*2", 6*2]
```

```
3. python [][:] == []
```

Highlighting **doesn't** work in Jupyter neither in Sphinx:

Three quotes, single line

```
1. python [2, 3, 1] != ["2", 3, 1]
```

```
2. python [4, 8, 12] == [2*2, "4*2", 6*2]
```

```
3. python [][:] == "[]"
```

Single quote, single line

```
1. python [2,3,1] != ["2",3,1]
2. python [4,8,12] == [2*2,"4*2",6*2]
3. python [][:] == "[]"
```

3.10 Toggable cells

There are various ways to have toggable cells, for example for showing exercise solutions. Some of the solutions just don't work. Preferred solution for now is with Javascript

3.10.1 Toggling with Javascript (PREFERRED)

- Works in Markdown
- Works while in Jupyter
- Works in HTML
- Does not show in Latex (which might be a good point, if you intend to put somehow solutions at the end of the document)
- NOTE: after creating the text to see the results you have to run the initial cell with `jupman.init` (as for the toc)
- NOTE: you can't use Markdown block code since of Sept 2017 doesn't show well in HTML output

3.11 HTML details in Markdown, code tag

- Works while in Jupyter
- Doesn't work in HTML output
- as of Sept Oct 2017, not yet supported in Microsoft browsers

[Click here to see the code](#)

```
question = raw_input("What?")
answers = random.randint(1,8)
if question == "":
    sys.exit()
```

3.11.1 HTML details in Markdown, Markdown mixed code

- Works while in Jupyter
- Doesn't work in HTML output
- as of Sept Oct 2017, not yet supported in Microsoft browsers

[Click here to see the code](#)

```
question = raw_input("What?")
answers = random.randint(1,8)
if question == "":
    sys.exit()
```

3.11.2 HTML details in HTML, raw NBConvert Format

- Doesn't work in Jupyter
- Works in HTML output
 - NOTE: as of Sept Oct 2017, not yet supported in Microsoft browsers
- Doesn't show at all in PDF output

Some other Markdown cell afterwards

3.12 Files in templates

Since Dec 2019 they are not accessible [see issue 10⁴⁸](#), but it is not a great problem, you can always put a link to Github, see for example [exam-yyyy-mm-dd.ipynb⁴⁹](#)

3.13 Python tutor

There are various ways to embed Python tutor, first we put the recommended one.

3.13.1 jupman.pytut

RECOMMENDED: You can put a call to `jupman.pytut()` at the end of a cell, and the cell code will magically appear in python tutor in the output (except the call to `pytut()` of course). Does not need internet connection.

```
[2]: x = [5,8,4,10,30,20,40,50,60,70,20,30]
     y= {3:9}
     z = [x]
```

```
jupman.pytut()
```

```
[2]: <IPython.core.display.HTML object>
```

jupman.pytut scope: BEWARE of variables which were initialized in previous cells, they WILL NOT be available in Python Tutor:

```
[3]: w = 8
```

```
[4]: x = w + 5
     jupman.pytut()
```

⁴⁸ <https://github.com/DavidLeoni/jupman/issues/10>

⁴⁹ https://github.com/DavidLeoni/jupman/tree/master/_templates/exam/exam-yyyy-mm-dd.ipynb

```
Traceback (most recent call last):
  File "/home/da/Da/prj/jupman/prj/jupman.py", line 2305, in _runscript
    self.run(script_str, user_globals, user_globals)
  File "/usr/lib/python3.5/bdb.py", line 431, in run
    exec(cmd, globals, locals)
  File "<string>", line 2, in <module>
NameError: name 'w' is not defined
```

```
[4]: <IPython.core.display.HTML object>
```

jupman.pytut window overflow: When too much right space is taken, it might be difficult to scroll:

```
[5]: x = [3,2,5,2,42,34,2,4,34,2,3,4,23,4,23,4,2,34,23,4,23,4,23,4,23,4,234,34,23,4,23,4,23,4,2]
jupman.pytut()
```

```
[5]: <IPython.core.display.HTML object>
```

```
[6]: x = w + 5
jupman.pytut()
```

```
Traceback (most recent call last):
  File "/home/da/Da/prj/jupman/prj/jupman.py", line 2305, in _runscript
    self.run(script_str, user_globals, user_globals)
  File "/usr/lib/python3.5/bdb.py", line 431, in run
    exec(cmd, globals, locals)
  File "<string>", line 2, in <module>
NameError: name 'w' is not defined
```

```
[6]: <IPython.core.display.HTML object>
```

jupman.pytut execution: Some cells might execute in Jupyter but not so well in Python Tutor, due to [its inherent limitations](#)⁵⁰:

```
[7]: x = 0
for i in range(10000):
    x += 1
print(x)
jupman.pytut()
```

```
10000
```

```
[7]: <IPython.core.display.HTML object>
```

jupman.pytut infinite loops: Since execution occurs first in Jupyter and then in Python tutor, if you have an infinite loop no Python Tutor instance will be spawned:

```
while True:
    pass

jupman.pytut()
```

jupman.pytut() resizability: long vertical and horizontal expansion should work:

```
[8]: x = {0:'a'}
for i in range(1,30):
    x[i] = x[i-1]+str(i*10000)
jupman.pytut()
```

⁵⁰ <https://github.com/pgbovine/OnlinePythonTutor/blob/master/unsupported-features.md>

```
[8]: <IPython.core.display.HTML object>
```

jupman.pytut cross arrows: With multiple visualizations, arrows shouldn't cross from one to the other even if underlying script is loaded multiple times (relates to `visualizerIdOverride`)

```
[9]: x = [1,2,3]
```

```
jupman.pytut()
```

```
[9]: <IPython.core.display.HTML object>
```

jupman.pytut print output: With only one line of print, Print output panel shouldn't be too short:

```
[10]: print("hello")
```

```
jupman.pytut()
```

```
hello
```

```
[10]: <IPython.core.display.HTML object>
```

```
[11]: y = [1,2,3,4]
```

```
jupman.pytut()
```

```
[11]: <IPython.core.display.HTML object>
```

3.13.2 HTML magics

Another option is to directly paste Python Tutor iframe in the cells, and use Jupyter `%%HTML` magics command.

HTML should be available both in notebook and website - of course, requires an internet connection.

Beware: you need the HTTPS !

```
[12]: %%HTML
```

```
<iframe width="800" height="300" frameborder="0"
      src="https://pythontutor.com/iframe-embed.html#code=x+%3D+5%0Ay+%3D+10%0Az+
      ↪+%3D+x+%2B+y&cumulative=false&py=2&curInstr=3">
</iframe>
```

```
<IPython.core.display.HTML object>
```

3.13.3 NBTutor

To show Python Tutor in notebooks, there is already a jupyter extension called [NBTutor](https://github.com/Igpage/nbtutor)⁵¹, afterwards you can use magic `%%nbtutor` to show the interpreter.

Unfortunately, it doesn't show in the generated HTML :-/

```
[13]: %reload_ext nbtutor
```

⁵¹ <https://github.com/Igpage/nbtutor>

```
[14]: %%nbtutor

for x in range(1,4):
    print("ciao")
x=5
y=7
x +y
```

3.14 Stripping answers

For stripping answers examples, see *jupyter-example/jupyter-example-sol*. For explanation, see *usage*

3.15 Metadata to HTML classes

3.16 Formatting problems

3.16.1 Characters per line

Python standard for code has limit to 79, many styles have 80 (see [Wikipedia](https://en.wikipedia.org/wiki/Characters_per_line)⁵²)

We can keep 80:

```
-----
-----
```

Errors hold 75 dashes:

Plain:

```
-----
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-15-9e1622b385b6> in <module>()
----> 1 1/0

ZeroDivisionError: division by zero
```

As Python markup:

```
-----
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-15-9e1622b385b6> in <module>()
----> 1 1/0

ZeroDivisionError: division by zero
```

⁵² https://en.wikipedia.org/wiki/Characters_per_line

On website this **may** display a scroll bar, because it will actually print ' apexes plus the dashes

```
[16]: '-'*80
```

```
[16]: '-----'
```

This should **not** display a scrollbar:

This should **not** display a scrollbar:

```
[18]: print('-'*80)
```

3.16.2 Very large input

In Jupyter: default behaviour, show scrollbar

On the website: should expand in horizontal as much as it wants, the rationale is that for input code since it may be printed to PDF you should always manually put line breaks.

[illegible]

(continued from previous page)

Very long HTML (and long code line)

Should expand in vertical as much as it wants.

```
[20]: %%HTML

<iframe width="100%" height="1300px" frameborder="0" src="https://umap.openstreetmap.
↪fr/en/map/mia-mappa-agritur_182055?scaleControl=false&miniMap=false&
↪scrollWheelZoom=false&zoomControl=true&allowEdit=false&moreControl=true&
↪searchControl=null&tilelayersControl=null&embedControl=null&datalayersControl=true&
↪onLoadPanel=undefined&captionBar=false#11/46.0966/11.4024"></iframe><p><a href=
↪"http://umap.openstreetmap.fr/en/map/mia-mappa-agritur_182055">See full screen</a></
↪p>

<IPython.core.display.HTML object>
```

3.16.3 Very long output

In Jupyter: by clicking, you can collapse

On the website: a scrollbar should appear

```
[21]: for x in range(150):
      print('long output ...', x)

long output ... 0
long output ... 1
long output ... 2
long output ... 3
long output ... 4
long output ... 5
long output ... 6
long output ... 7
long output ... 8
long output ... 9
long output ... 10
long output ... 11
long output ... 12
long output ... 13
long output ... 14
long output ... 15
long output ... 16
long output ... 17
long output ... 18
long output ... 19
long output ... 20
long output ... 21
long output ... 22
long output ... 23
long output ... 24
long output ... 25
```

(continues on next page)

(continued from previous page)

```
long output ... 26
long output ... 27
long output ... 28
long output ... 29
long output ... 30
long output ... 31
long output ... 32
long output ... 33
long output ... 34
long output ... 35
long output ... 36
long output ... 37
long output ... 38
long output ... 39
long output ... 40
long output ... 41
long output ... 42
long output ... 43
long output ... 44
long output ... 45
long output ... 46
long output ... 47
long output ... 48
long output ... 49
long output ... 50
long output ... 51
long output ... 52
long output ... 53
long output ... 54
long output ... 55
long output ... 56
long output ... 57
long output ... 58
long output ... 59
long output ... 60
long output ... 61
long output ... 62
long output ... 63
long output ... 64
long output ... 65
long output ... 66
long output ... 67
long output ... 68
long output ... 69
long output ... 70
long output ... 71
long output ... 72
long output ... 73
long output ... 74
long output ... 75
long output ... 76
long output ... 77
long output ... 78
long output ... 79
long output ... 80
long output ... 81
long output ... 82
```

(continues on next page)

(continued from previous page)

```
long output ... 83
long output ... 84
long output ... 85
long output ... 86
long output ... 87
long output ... 88
long output ... 89
long output ... 90
long output ... 91
long output ... 92
long output ... 93
long output ... 94
long output ... 95
long output ... 96
long output ... 97
long output ... 98
long output ... 99
long output ... 100
long output ... 101
long output ... 102
long output ... 103
long output ... 104
long output ... 105
long output ... 106
long output ... 107
long output ... 108
long output ... 109
long output ... 110
long output ... 111
long output ... 112
long output ... 113
long output ... 114
long output ... 115
long output ... 116
long output ... 117
long output ... 118
long output ... 119
long output ... 120
long output ... 121
long output ... 122
long output ... 123
long output ... 124
long output ... 125
long output ... 126
long output ... 127
long output ... 128
long output ... 129
long output ... 130
long output ... 131
long output ... 132
long output ... 133
long output ... 134
long output ... 135
long output ... 136
long output ... 137
long output ... 138
long output ... 139
```

(continues on next page)

(continued from previous page)

```
long output ... 140
long output ... 141
long output ... 142
long output ... 143
long output ... 144
long output ... 145
long output ... 146
long output ... 147
long output ... 148
long output ... 149
```

[]:

CHAPTER
FOUR

CHAPTER EXAMPLES

5.1 Python example

Example of notebook for exercises in Python files

5.1.1 Download exercises zip

Browse files online⁵³

5.1.2 What to do

- unzip exercises in a folder, you should get something like this:

```
python-example
python-example.ipynb
lab1.py
lab1_test.py
lab1_sol.py
lab2.py
lab2_test.py
lab2_sol.py
jupman.py
my_lib.py
```

- open the editor of your choice (for example Visual Studio Code, Spyder or PyCharme), you will edit the files `lab1.py` and `lab2.py`
- Go on reading this notebook, and follow instuctions inside.

⁵³ <https://github.com/DavidLeoni/jupman/tree/master/python-example>

Let's begin

You are going to program a simulator of bouncing clowns. To do so, we are going to load this module:

```
[2]: import local
```

```
[3]: local.gimme(5)  
It was a 5 indeed
```

Download test data

Local file:

- example.txt
- example.csv

5.1.3 Global image



5.1.4 Local exercise image



5.1.5 Python tutor

```
[4]: x = [1,2,3]
     y = 6

     jupman.pytut()
[4]: <IPython.core.display.HTML object>
```

```
[5]: y = [1,2,3]

     jupman.pytut()
[5]: <IPython.core.display.HTML object>
```

Start editing lab1.py in text editor

```
[6]: from lab1_sol import *
```

5.1.6 add

Implement add function:

```
[7]: add(3,5)
[7]: 8
```

5.1.7 sub

Implement sub function

```
[8]: sub(7,4)
[8]: 3
```

```
[ ]:
```

5.2 Jupyter example

5.2.1 Download exercises zip

Browse files online⁵⁴

Example of notebook for exercises in Jupyter files.

For python files based example and more, see *Python example*

⁵⁴ <https://github.com/DavidLeoni/jupman/tree/master/jupyter-example>

5.2.2 What to do

- unzip exercises in a folder, you should get something like this:

```
jupyter-example
  jupyter-example.ipynb
  jupyter-example-sol.ipynb
  jupman.py
  my_lib.py
```

WARNING: to correctly visualize the notebook, it **MUST** be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `jupyter-example/jupyter-example.ipynb`
- Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press `Control + Enter`
- to execute Python code inside a Jupyter cell AND select next cell, press `Shift + Enter`
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press `Alt + Enter`
- If the notebooks look stuck, try to select Kernel -> Restart

```
[2]: # REMEMBER TO IMPORT jupman !
      # This cell needs to be executed only once, you can usually find it at the beginning
      ↪ of the worksheets

      import jupman
```

```
[3]: x = [1,2,3]
      y = x
      jupman.pytut()

[3]: <IPython.core.display.HTML object>
```

```
[4]: y = [1,2,3]
      w = y[0]
      jupman.pytut()

[4]: <IPython.core.display.HTML object>
```

5.2.3 Exercise 1

Implement `inc` function:

```
<div id="jupman-sol-7" class="jupman-sol jupman-sol-code" onclick="jupman.toggleSolution('jupman-sol-7');">
  <input type="button" class="jupman-sol-label" data-jupman-show="Show solution" data-jupman-hide="Hide"
  value="Show solution"/> <div class="jupman-sol-content" style="display:none">
```

```
[5]:
```

(continues on next page)

(continued from previous page)

```
def helper(x):
    return x + 1

def inc(x):

    return helper(x)
```

```
</div> </div>
```

```
[5]: def inc(x):
      raise Exception('TODO IMPLEMENT ME !')
```

5.2.4 Exercise 2

Implement upper function

```
<div id="jupman-sol-9" class="jupman-sol jupman-sol-code" onclick="jupman.toggleSolution('jupman-sol-9');">
> <input type="button" class="jupman-sol-label" data-jupman-show="Show solution" data-jupman-hide="Hide"
value="Show solution"/> <div class="jupman-sol-content" style="display:none">
```

```
[6]: def helper2(x):
      return x.upper()

def upper(x):

    return helper2(x)
```

```
</div> </div>
```

```
[6]: def upper(x):
      raise Exception('TODO IMPLEMENT ME !')
```

Exercise 3

Note everything *after* the ‘write here’ comment will be discarded. Note you can put how many spaces you want in the comment

```
<div id="jupman-sol-11" class="jupman-sol jupman-sol-code" onclick="jupman.toggleSolution('jupman-sol-11');">
> <input type="button" class="jupman-sol-label" data-jupman-show="Show solution" data-jupman-hide="Hide"
value="Show solution"/> <div class="jupman-sol-content" style="display:none">
```

```
[7]: w = 5

# write here
```

(continues on next page)

(continued from previous page)

```
x = 5 + 6
y = 6.4
z = x / y
```

```
</div> </div>
```

[7]:

```
w = 5

# write here
```

Exercise 4

Shows how to completely remove the content of a solution cell (including the solution comment)

EXERCISE: write a function that prints 'hello'

```
<div id="jupman-sol-14" class="jupman-sol jupman-sol-code" onclick="jupman.toggleSolution('jupman-sol-14');"
> <input type="button" class="jupman-sol-label" data-jupman-show="Show solution" data-jupman-hide="Hide"
value="Show solution"/> <div class="jupman-sol-content" style="display:none">
```

[8]: # SOLUTION

```
def f():
    print('hello')
```

```
</div> </div>
```

[8]:

Exercise 5

Shows the QUESTION / ANSWER feature. All content in 'ANSWER:' cell will be stripped

QUESTION: Describe why iPhone n + 1 is better than iPhone n

```
<div id="jupman-sol-17" class="jupman-sol jupman-sol-question" onclick="jupman.toggleSolution('jupman-sol-17');"
> <input type="button" class="jupman-sol-label" data-jupman-show="Show answer" data-jupman-hide="Hide"
value="Show answer"/> <div class="jupman-sol-content" style="display:none">
```

ANSWER: it costs more

```
</div> </div>
```

5.2.5 Conclusion

bla bla

Relative image test, Markdown format:



Relative image test, HTML `img` tag:



Relative link test, Markdown format:

Back to index

Relative link test, HTML `a` tag:

Back to index

[]:

5.3 Jupyter and python example

Example of notebook for exercises both in Jupyter and Python files

5.3.1 Download exercises zip

Browse files online⁵⁵

5.3.2 What to do

- unzip exercises in a folder, you should get something like this:

```
jup-and-py-example
jup-and-py-example.ipynb
jup-and-py-example_sol.ipynb
lab.py
lab_test.py
lab_sol.py
```

- open the editor of your choice (for example Visual Studio Code, Spyder or PyCharme), and edit `lab.py` file
- Go on reading this notebook, and follow instructions inside.

⁵⁵ <https://github.com/DavidLeoni/jupman/tree/master/jup-and-py-example>

Let's begin

You are going to program a simulator of bouncing clowns. To do so, we are going to load this module:

```
[2]: import local
```

```
[3]: local.gimme(5)  
It was a 5 indeed
```

Download test data

Local file:

- example.txt
- example.csv

5.3.3 Global image



5.3.4 Local exercise image



5.3.5 Python tutor

```
[4]: x = 5
      y = 6
      z = x + y

      jupman.pytut()

[4]: <IPython.core.display.HTML object>
```

Exercise in Jupyter

Implement this function:

```
<div id="jupman-sol-12" class="jupman-sol jupman-sol-code" onclick="jupman.toggleSolution('jupman-sol-12');">
  <input type="button" class="jupman-sol-label" data-jupman-show="Show solution" data-jupman-hide="Hide"
  value="Show solution"/> <div class="jupman-sol-content" style="display:none">
```

```
[5]: def hello(s):

      print(s)

      hello("Guybrush")

      Guybrush
```

```
</div> </div>
```

```
[5]: def hello(s):
      raise Exception('TODO IMPLEMENT ME !')

      hello("Guybrush")

      Guybrush
```

Start editing lab.py in text editor

```
[6]: from lab_sol import *
```

5.3.6 add

Implement add function:

```
[7]: add(3,5)

[7]: 8
```

5.3.7 sub

Implement sub function

```
[8]: sub(7, 4)
```

```
[8]: 3
```

```
[ ]:
```


TEMPLATES

6.1 Changelog

Jupman Jupyter Manager

jupman.softpython.org⁵⁶

6.1.1 October 16 2020 - 3.2

- added optional build on Github Actions
- solutions are finally hidden on the website, with a click to show button!
- moved to jupman.softpython.org
- updated nbsphinx to 0.7.1
- updated sphinx_rtd_theme to 0.4.3
- updated sphinx to 2.3.1
- updated pygments to 2.7.1

6.1.2 January 16th 2020 - 3.1

- removed `jupman.init` root parameter
- bugfixes
- upgraded nbsphinx from 0.3.4 to 0.5.0
- upgraded sphinx_rtd_theme from 0.2.5b1 to 0.4.3
- upgraded sphinx from 1.7.6 to 2.3.1
- upgraded recommonmark from 0.4.0 to 0.6.0

⁵⁶ <https://jupman.softpython.org>

6.1.3 December 29th 2019 - 3.0

- much simplified folder structure
 - [Issue 33](#)⁵⁷
- removed solutions from header requirement
 - [Issue 32](#)⁵⁸
- introduced tests (pytest, hypothesis)
- removed old_news in favor of changelog.md
- Latex:
 - much better PDF cover
 - using xelatex
 - set up unicode mappings
- several fixes

6.1.4 September 24th 2018 - 2.0

- now using index.ipynb as home. Hurray !

6.1.5 September 19th 2018 - 1.0

- fixed build.py
- added html templates examples
- cleaned toc (was showing too much when loading)

6.1.6 August 26th 2018 - 0.9

- implemented generation of exercises from solutions [Issue 14(<https://github.com/DavidLeoni/jupman/issues/14>)
- reverted to old jupman.init() code [Issue 12](#)⁵⁹

6.1.7 August 12th 2018 - 0.8

- Prepended all functions in jupman.py with jupman_
- replaced index with proper homepage. see [Issue 11](#)⁶⁰
 - from now on you need home.ipynb file, because replacing index.rst is a nightmare!
 - new index.rst is just a placeholder which simply redirects to home.html. Do not modify it.
 - put the toctree in toc.rst

⁵⁷ <https://github.com/DavidLeoni/jupman/issues/33>

⁵⁸ <https://github.com/DavidLeoni/jupman/issues/32>

⁵⁹ <https://github.com/DavidLeoni/jupman/issues/12>

⁶⁰ <https://github.com/DavidLeoni/jupman/issues/11>

- exercises ipynb can now stay in exercises/ folder; when exercises are zipped, jupman automatically adds to the zip the required site files. see [Issue 12](#)⁶¹
- Tried %run at beginning of notebooks, without much satisfaction (see discussion in [Issue 12](#)⁶²):
- disabled toc by default in html files. To enable it, in python use %run -i ../../jupman --toc
- renamed past-exams directory from 'past-exams' to 'exams'
- created info, error, warn, fatal functions to conf.py
- introduced new variable exercise_common_files in conf.py for common files to be zipped
- added pages exam-project, markdown, project-ideas,
- added cc-by.png
- renamed changelog.txt to changelog.md
- now using templates with curly brackets in templating, like _JM_{some_property}
- jupman.js : now when manually saving html in Jupyter, resulting html correctly hides cells
- Fixes <https://github.com/DavidLeoni/jupman/issues/2> : now toc is present in local build for pdfs

6.1.8 August 3rd 2018 - 0.7

- added jupman.py pytut() for displaying Python tutor in the cells
- added jupman.py toc=False option to jupman.py init to disable toc
- removed jupman.pyuseless networkx import from
- fixed usage indentation
- added changelog.txt

6.2 Past Exams

[]:

6.3 Exam project

For general (credits, attendance), see course description at section Evaluation and exams

Delivery times

Ideas for possible projects: [See here](#)

Last update: TODO

In short:

⁶¹ <https://github.com/DavidLeoni/jupman/issues/12>

⁶² <https://github.com/DavidLeoni/jupman/issues/12>

6.3.1 What to do

First of all: send by email to TODO@TODO.COM a brief description of the project, to decide what to do. I will create a Google doc to keep track of progresses and / or problems found.

Once the project is defined, go on like this:

1 - Download [zip with template](#) (view online files [TODO](#)⁶³)

After unzipped, you will find a folder named NAME-SURNAME-ID, with these files inside:

```
- NAME-SURNAME-ID
  - project.ipynb
  - markdown.ipynb
  - requirements.txt
  - img
    - example.png
```

2 - Rename the folder NAME-SURNAME-ID with your data

3 - run Jupyter from the folder you just renamed

4 - edit file `project.ipynb`, **closely following the indications** *in the following technical requirements*

5 - Once done, send project by email to TODO@TODO.COM

6.3.2 Technical requirements

Write in *Markdown*

Python code

`requirements.txt` file

Graphical interfaces

Be careful to

6.4 Project ideas

6.4.1 TODO

Last update: TODO

⁶³ <https://www.GITHUB.TODO>

6.4.2 Introduction

[]:

6.5 Jupman Project

METTERE:

TITOLO

NOME - MATRICOLA

DATA

6.5.1 Introduzione

Descrivere

- gli obbiettivi che ci si è posti
- risultati attesi (i.e. file CSV con dati integrati da mostrare su Umap)
- librerie Python utilizzate per il progetto

6.5.2 Sorgenti dati

Descrivere le sorgenti dati (almeno due), mettendo:

- nome origine e possibilmente URL
- le licenze
- encoding
- dimensione in Kb (i file dovrebbero essere max 50 megabyte, se più grandi cercate un modo per ridurli e se avete problemi chiedete come fare)
- schemi dei dati
 - CSV: intestazione colonne, tipi delle colonne (stringa, numero, data ...)
 - JSON: spiegare sommariamente le scheletro del JSON (se volete strafare scrivete un [JSON schema](http://json-schema.org)⁶⁴, ma non l'abbiamo visto a lezione)
 - XML: spiegare sommariamente le scheletro dell'XML (possibilmente guardate se l'XML già fornisce un [XML schema](https://en.wikipedia.org/wiki/XML_schema)⁶⁵, ma non l'abbiamo fatto a lezione, se avete dubbi chiedete)
 - SQL: mettere schema del DB (molti browser di database possono generare diagrammi per gli schemi)
 - per altri formati: descrivere a parole
- qualche dato di esempio
- Se i dati sono da ottenere tramite [web API](https://en.wikipedia.org/wiki/Web_API)⁶⁶, chiedersi se è possibile ottenere gratuitamente tutti i dati desiderati entro i limiti d'uso dell'API

⁶⁴ <http://json-schema.org>

⁶⁵ https://en.wikipedia.org/wiki/XML_schema

⁶⁶ https://en.wikipedia.org/wiki/Web_API

IMPORTANTE: ricordarsi di includere nella cartella del progetto una copia dei dati! Questa è fondamentale ai fini della riproducibilità del notebook, e vale in particolare per i dati ottenuti da Web API e pagine HTML, che possono variare nel tempo (i.e. dati meteo).

6.5.3 Pulizia e integrazione dati

Elencare eventuali problemi da correggere, e come li si è risolti. Esempi:

- dati mal formattati
- interpretazione dati mancanti, come li si è convertiti
 - i valori assenti sono da assumersi uguali a 0, stringa vuota, `None`, lista vuota ? ...
- Mappe con punti in posti assurdi

6.5.4 Analisi

Mettere qualche analisi sui dati. Esempi:

- mostrare raggruppamenti, grafici frequenze
- correlazioni tra valori con Pandas
- mostrare punti su mappa con UMap, notare raggruppamenti
- ricavare qualche modello dai dati, con plot su grafico(es: relazione lineare pressione / temperatura)
- mettere qualche widget per variare parametri del modello

6.5.5 Problematiche riscontrate

Descrivere i problemi maggiori riscontrati nel progetto

6.5.6 Conclusioni

Descrivere:

- se si sono raggiunti gli obiettivi posti
- eventualmente cosa si potrebbe aggiungere di interessante

6.6 Markdown

Per formattare il testo, Jupyter mette a disposizione un linguaggio chiamato Markdown. Perché dovreste imparare Markdown? E' semplice, molto popolare ed è probabile trovarlo in molti posti (blog, sistemi di documentazione tecnica, etc).

Qua riporto solo informazioni essenziali, per altre potete consultare la [Guida di Jupyter \(inglese\)](http://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%20Cells.html)⁶⁷

⁶⁷ <http://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%20Cells.html>

6.6.1 Celle Markdown

Per dire a Jupyter che una cella è codice Markdown e non Python, dal menu seleziona `Cell->Cell type->Markdown`. Una shortcut veloce è premere `Esc` seguito poi da il tasto `m`

6.6.2 Paragrafi

Per suddividere paragrafi basta inserire una riga vuota:

Per esempio, scrivendo così:

```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
↳incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
↳exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu
↳fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
↳culpa qui officia deserunt mollit anim id est laborum.

```

Si ottiene questo:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

6.6.3 Link

Un link si scrive mettendo il testo visibile dall'utente tra parentesi quadre e l'indirizzo vero e proprio tra parentesi tonde:

```
[Questo è il testo del link](http://www.google.com)
```

Il risultato sarà il seguente:

Questo è il testo del link⁶⁸

6.6.4 Liste

Le liste si scrivono mettendo anteposendo ad ogni elemento un asterisco:

Per esempio, scrivendo questo:

```

* Elemento 1
* Elemento 2
  * Sotto elemento
  * Altro sotto elemento
* Elemento 3

```

Verrà visualizzato così:

- Elemento 1
- Elemento 2

⁶⁸ <http://www.google.com>

- Sotto elemento
- Altro sotto elemento
- Elemento 3

Potete anche usare liste numerate antepoendo agli elementi un numero e un punto.

Per esempio, questo:

```
1. Elemento 1
2. Elemento 2
  1. Sotto elemento
  2. Altro sotto elemento
3. Elemento 3
```

Verrà visualizzato così:

1. Elemento 1
2. Elemento 2
 1. Sotto elemento
 2. Altro sotto elemento
3. Elemento 3

6.6.5 Immagini

Purtroppo Jupyter non supporta il copia e incolla di immagini, potete solo inserire dei link alle immagini stesse da mettere nella stessa cartella del notebook oppure in una sottocartella.

Una volta che la cella viene eseguita, se il percorso al file è corretto apparirà l'immagine. Per indicare il percorso, scrivete punto esclamativo, parentesi quadre aperta-chiusa, e poi tra parentesi tonde il percorso del file, per es se l'immagine `notebook_icon.png` sta nella sotto-cartella `img`, scrivete così:

```
![] (img/example.png)
```

Eseguendo la cella, apparirà l'immagine:



Nota che potete usare qualunque formato di immagine (`jpg`, `png`, `bmp`, `svg`, per gli altri provate a vedere se vengono visualizzati).

6.6.6 Variabili e nomi tecnici

Per visualizzare in evidenza variabili e nomi tecnici, come `x`, `faiQualcosa`, `percorso-di-file`, potete includere il nome tra due cosiddetti backtick ```

NOTA: il backtick ``` NON è l'apice che usiamo di solito: `'`

Per scrivere questo strano apice rovesciato, guardate qua, se non va fate copia e incolla !

- Windows:
 - se avete il tastierino numerico: tenere premuto `Alt` `Gr`, scrivere `96` sul tastierino numerico, rilasciare `Alt` `Gr`
 - se non l'avete: provate a premere tasto windows + `\` (in alto a sinistra)
- Mac: `alt+9`
- Linux: `Alt-Gr` + carattere apice normale `'`

6.6.7 Codice JSON / XML / Python

Se in una cella Markdown volete visualizzare testo posizionato esattamente come lo scrivete, racchiudetelo in un blocco delimitato da file di tre tre backtick ````` :

```

```
testo posizionato come
 voglio
io
```

```

Risultato:

```
testo posizionato come
    voglio
io
```

Il codice python / json / xml e altri possono essere formattati automaticamente da Jupyter. Basta scriverlo in blocchi da tre backtick come prima e in più specificare il linguaggio subito dopo i primi tre backtick, per esempio un json scritto così:

```json

```
{
 "a" : ["b"],
 "c" : {
 "d":5
 }
}
```

```

Risulterà formattato in questo modo:

```
{
  "a" : ["b"],
  "c" : {
```

(continues on next page)

(continued from previous page)

```
"d": 5
}
}
```

6.6.8 Formule matematiche

E' possibile scrivere formule in [Latex](#)⁶⁹ (ma non l'abbiamo visto a lezione) mettendole tra segni di dollaro \$. Per esempio \sum verrà visualizzato così: \sum

Tabelle

Scrivere tabelle piccole in Markdown è ancora fattibile:

Per esempio, scrivere questo:

```
io |sono | una tabella
---|-----|-----
4 |ciao|3
2 |hello world|7
```

risulta visualizzato così :

io	sono	una tabella
4	ciao	3
2	hello world	7

ma per tabelle grandi Markdown è terribile. Quindi siete più che giustificati a usare alternative, per esempio allegare fogli Excel (anche se preferisco LibreOffice Calc col formato `.ods`, se proprio amate Excel perlomeno mandatemi un `.xls` e non un `.xlsx`). Potete anche prendere screenshot delle tabelle e includerli come immagini.

⁶⁹ http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf#chapter.5

CHAPTER
SEVEN

INDEX