NBBinder

Jul 06, 2020

Contents:

THIS PACKAGE HAS BEEN ARCHIVED DUE TO NAMING COLLISION. It is survived by the package NBJoint.

NBBinder generates a navigable book-like structure to a collection of Jupyter notebooks.

CHAPTER 1

Overview

NBBinder generates a navigable book-like structure for a collection of Jupyter notebooks.

1.1 Description

The main function in this module is called `bind()`. It reads a collection of Jupyter notebooks from a given directory and, upon configuration,

- adds a **table of contents** to a selected notebook file, with links to the other notebooks;
- adds a **header** cell to each notebook, with custom information about the collection of notebooks;
- adds a **badge** cell to each notebook, with links to opening the notebooks in different platforms or formats. For instance, on can include a **Google Colab badge** and a **Binder badge**, with links to opening each notebook in these cloud computing plataforms, a badge for showing **slides** as exported with `nbconvert`, and so on.
- adds **navigator links**, at the beggining and at the end of each notebook, with links to traverse to the previous and the next notebook, and to other selected notebooks, such as the Table of Contents and the References;
- **exports** the notebooks to other formats using `nbconvert`, so that, for example, **slides** can be generated automatically and in bulk.

1.2 Functions

The function `bind()` can be called in two different ways:

- *directly with the arguments* to be applied in the bindind process; or
- *with a configuration file* as argument, with the configuration file containing the desired arguments.

The `bind()` function calls the following functions in this module, which take care of each of the main features of the notebook binder:

- `reindex()`: reorder the notebooks when a new notebook is to be inserted between others or whether there are gaps in the indices;
- `add_contents()`: adds the Table of Contents to a selected "Contents" file;
- `add_headers()`: adds a header to each notebook with a given custom information;
- `add_badges()`: adds a badge cell to each notebook with one or more badges to open up the document in different platforms or formats;
- `add_navigators()`: adds navigation bars to the top and bottom of each notebook.
- `export_notebooks()`: exports the notebooks to any of the different formats as provided by nbconvert: HTML, LaTeX, PDF, Reveal JS, Markdown (md), ReStructured Text (rst), executable script. Notice that `add_badges()` can be used to link to the exported notebooks, useful, for instance, to access slides of the notebooks for presentation in class.

Each of these later functions can be called separately, if only some of these features are desired.

When running `nbbinder.py` as a script, it expects the filename of the configuration file and calls the function `bind(config_file)`, where config_file is the name of the configuration file.

Look at the documentation for more information on each of these functions and for the other functions available on this package.

1.3 Examples

1.3.1 A little taste

For instance, upon proper configuration (see section *Notebooks with slides and cloud computing badges* below), the collection of bare notebooks in the folder Water bare collection is bound to the folder Water bound collection, and, in particular, the file 00.00-Water_Contents.ipynb receives a table of contents, a header, navigator cells and the badges

Below we show some examples in more details.

1.3.2 Example with configuration file

The most convenient way to use the module, or script, is via a configuration file. The configuration files are written in the YAML format.

For instance, consider the following `config_nb_alice.yml`, which is included in the `tests` folder of the repository:

```
# Configuration file for the python module NBBinder

version: 0.13a

path_to_notes: nb_builds/nb_alice

contents:
  toc_nb_name: 00.00-Alice's_Adventures_in_Wonderland.ipynb
  toc_title: Table of Contents
  show_index_in_toc: True

header: "NBBinder test on a collection of notebooks named after the chapters of 'Alice
↪'s Adventures in Wonderland'"
```

(continues on next page)

(continued from previous page)

```
navigators:
  core_navigators:
    - 00.00-Alice's_Adventures_in_Wonderland.ipynb
  show_nb_title_in_nav: False
  show_index_in_nav: False
```

Then, we import the module and use the `bind()` function with this configuration file as argument:

```
import nbbinder as nbb
nbb.bind('config_nb_alice.yml')
```

Or we execute it as a script in the command line:

```
./nbbinder.py config.yml
```

The key `path_to_notes` indicates that the notebooks are in the folder `nb_builds/nb_alice`, relative to where the script that calls the function `bind()` is located. In this folder, one finds the following notebooks, properly indexed:

```
00.00-Alice's_Adventures_in_Wonderland.ipynb
01.00-Down_the_Rabbit-Hole.ipynb
02.00-The_Pool_of_Tears.ipynb
03.00-A_Caucus-Race_and_a_Long_Tale.ipynb
04.00-The_Rabbit_Sends_in_a_Little_Bill.ipynb
05.00-Advice_from_a_Caterpillar.ipynb
06.00-Pig_and_Pepper.ipynb
07.00-A_Mad_Tea-Party.ipynb
08.00-The_Queen's_Croquet-Ground.ipynb
09.00-The_Mock_Turtle's_Story.ipynb
10.00-The_Lobster_Quadrille.ipynb
11.00-Who_Stole_the_Tarts?.ipynb
12.00-Alice's_Evidence.ipynb
```

The function `bind()` then reads the notebooks and *binds* them accordingly. In particular, the following table of contents is added to the file indicated by the key `toc_nb_name` in the configuration file:

```
Table of Contents
Alice's Adventures in Wonderland
1. Down the Rabbit-Hole
2. The Pool of Tears
3. A Caucus-Race and a Long Tale
4. The Rabbit Sends in a Little Bill
5. Advice from a Caterpillar
6. Pig and Pepper
7. A Mad Tea-Party
8. The Queen's Croquet-Ground
9. The Mock Turtle's Story
10. The Lobster Quadrille
11. Who Stole the Tarts?
12. Alice's Evidence
```

See 00.00-Alice's_Adventures_in_Wonderland.ipynb for the actual bound version of the first notebook. Notice the **header** in the begining of the notebook and the **navigator** cells after the header and at the end of the notebook. Experiment with the navigator links to move to the other notebooks.

1.3.3 Notebooks with subsections

By appropriately naming the files, we can have different formattings for the *Table of Contents*. For instance, if your list of files is

```
00.00-Front_Page.ipynb
01.00-Introduction.ipynb
02.00-Project_Requirements.ipynb
03.00-The_History_of_Grammar.ipynb
04.00-Parts_of_Speech.ipynb
04.01-Nouns.ipynb
04.02-Verbs.ipynb
04.03-Adjectives.ipynb
04.04-Adverbs.ipynb
05.00-Sentences.ipynb
05.01-Complex_Sentences.ipynb
05.02-Compound_Sentences.ipynb
06.00-Paragraphs.ipynb
06.01-Descriptive.ipynb
06.02-Expository.ipynb
06.03-Narrative.ipynb
06.04-Persuasive.ipynb
07.00-Conclusion.ipynb
A0.00-Appendix.ipynb
BA.00-Glossary.ipynb
BB.00-Bibliography.ipynb
BC.00-Index.ipynb
```

we get, with a suitable configuration, the *Table of Contents*

```
Table of Contents
Front Page
1. Introduction
2. Project Requirements
3. The History of Grammar
4. Parts of Speech
  4.1. Nouns
  4.2. Verbs
  4.3. Adjectives
  4.4. Adverbs
5. Sentences
  5.1. Complex Sentences
  5.2. Compound Sentences
6. Paragraphs
  6.1. Descriptive
  6.2. Expository
  6.3. Narrative
  6.4. Persuasive
7. Conclusion
A. Appendix
Glossary
Bibliography
Index
```

See 00.00-Front-Page.ipynb for the actual bound version of the first notebook.

The binder for the notebooks in this collection is configured to include *badges* to render, in nbviewer, either the Jupyter notebook itself or the exported version to markdown. The *badge* cell is located just below the header. Just click the badge with the **mouse right button** to open it. If clicking it with the right button, from within github, nothing will happen.

1.3.4 Notebooks with preheaders

This is particularly useful for lectures notes. For instance, by naming your collection of notebooks as

```
00.00-Introduction.ipynb
01.00.Lecture-Math_Background.ipynb
01.01-Vector_Calculus.ipynb
01.02-Rigid_Motions.ipynb
02.00.Lecture-Kinematics.ipynb
02.01.Lecture-Velocity_and_Acceleration.ipynb
02.02.Lecture-Different_Types_of_Motions_and_Their_Components.ipynb
03.00.Lecture-Dynamics.ipynb
03.01..Part-Force_and_Momentum.ipynb
03.02..Part-Orbits_of_Planets_and_Satellites.ipynb
03.03..Part-Interception_and_Rendezvous.ipynb
04.00.Lecture-Trajectory_Optimization.ipynb
04.01.Lecture.Part-Performance.ipynb
04.02.Lecture.Part-Gravity_Turn.ipynb
04.03.Lecture.Part-Optimization.ipynb
AA.00-References.ipynb
```

we get, with a suitable configuration, the *Table of Contents*

```
Contents
Introduction
Lecture 1. Math Background
  1.1. Vector Calculus
  1.2. Rigid Motions
Lecture 2. Kinematics
  Lecture 2.1. Velocity and Acceleration
  Lecture 2.2. Different Types of Motions and Their Components
Lecture 3. Dynamics
  Part 1. Force and Momentum
  Part 2. Orbits of Planets and Satellites
  Part 3. Interception and Rendezvous
Lecture 4. Trajectory Optimization
  Lecture 4. Part 1. Performance
  Lecture 4. Part 2. Gravity Turn
  Lecture 4. Part 3. Optimization
References
```

See 00.00-Introduction.ipynb for the actual bound version of the first notebook.

Notice, above, different forms of displaying the parts of the same lecture note.

The binder for the notebooks in this collection is configured to include a *badge* to open them in nbviewer. The *badge* is located just below the header. Just click the badge with the **mouse right button** to open it. If clicking it with the right button, from within github, nothing will happen.

1.4 Notebooks with slides and cloud computing badges

The following configuration file is used in the collection of files present in the folder Water:

```
# Configuration file for the python module NBBinder

version: 0.13a

path_to_notes: nb_builds/nb_water

contents:
  toc_nb_name: 00.00-Water_Contents.ipynb
  toc_title: Table of Contents
  show_index_in_toc: True

header: "[*NBBinder test on a collection of notebooks about some thermodynamic␣
↪properperties of water*](https://github.com/rmsrosa/nbbinder)"

navigators:
  core_navigators:
    - 00.00-Water_Contents.ipynb
    - BA.00-References.ipynb
  show_nb_title_in_nav: True
  show_index_in_nav: False

badges:
  - title: Open in Google Colab
    url: https://colab.research.google.com/github/rmsrosa/nbbinder/blob/master/tests/
↪nb_builds/nb_water
    src: https://colab.research.google.com/assets/colab-badge.svg
  - title: Open in binder
    url: https://mybinder.org/v2/gh/rmsrosa/nbbinder/master?filepath=tests/nb_builds/
↪nb_water
    src: https://mybinder.org/badge.svg
  - title: View in NBViewer
    url: https://nbviewer.jupyter.org/github/rmsrosa/nbbinder/blob/master/tests/nb_
↪builds/nb_water
    label: view in
    message: nbviewer
    color: orange
  - title: View Slides
    url: https://nbviewer.jupyter.org/github/rmsrosa/nbbinder/blob/master/tests/nb_
↪builds/nb_water_slides
    extension: .slides.html
    label: view
    message: slides
    color: darkgreen

exports:
  - export_path: nb_builds/nb_water_slides
    exporter_name: slides
    exporter_args:
      reveal_scroll: True
```

After binding the collection, the folder Water bound collection is created. See 00.00-Water_Contents.ipynb for the first notebook, containing the table of contents. Now, each notebook has a badge cell with badges to open the notebooks in Google Colab, Binder, and nbviewer, and a final badge to open the associated Reveal.JS slides.

For the slides, the folder Water Slides is created via nbconvert, in accordance to the parameters associated with the key `exports` in the configuration file.

The **badge cell** looks like

CHAPTER 2

Installation

The package can be installed from PyPi with

```
pip install nbbinder
```

It can also be downloaded directly from github.com/rmsrosa/nbbinder and installed, from the downloaded package directory, with

```
pip install .
```

If you do not wish to install the package, you can simply download it and import it as a local module as follows:

- If the subdirectory `nbbinder` of the project is in the same folder as the script that will import it, simply do

  ```
  import nbbinder as nb
  ```

- If the subdirectory `nbbinder` is in a different location, use

  ```
  import os
  import sys

  sys.path.insert(0, os.path.abspath(os.path.join(os.getcwd(), 'path', 'from',
   ↪'script', 'to', 'module')))

  import nbbinder as nbb
  ```

In case of downloading the package and using it or installing it locally, you just need the file `nbbinder.py` in the root directory‘.

CHAPTER 3

Usage

3.1 Numbering the collection of notebooks

NBBinder binds a collection of notebooks belonging to a specified directory.

In order to be processed, each notebook in the collection should start with a pair of *file numberings*, separated by a dot and followed by a dash.

Each file numbering is composed of two characters. We refer to each of the file numberings as `N1` and `N2`. Thus, a notebook should have the form

`N1.N2-notebookfilename.ipynb`.

Each file numbering should be of one of the following forms:

- Two digits, from `00` to `99`;
- An uppercase letter followed by a digit, from `A0` to `Z9`;
- Two uppercase letters, from `AA` to `ZZ`.

Each file numbering is translated into a *head numbering*, for display in the table of contents and in the navigators.

The file numberings `N1` and `N2` are two hierarchical levels for the headings, such as "Chapter" and "Section", or "Section" and "Subsection".

The translation from file to heading numbering can be summarized in the following table:

`00` => empty string

`01` to `09` => `1` to `9`

`10` to `99` => `10` to `99`

`A0` to `Z0` => `A` to `Z`

`A1` to `Z9` => `A1` to `Z9`

`AA` to `ZZ` => empty string

Notice that the file numbering `00` and the pure alphanumeric numberings `AA` to `ZZ` lead to an empty string, which means no heading numbering is shown in the table of contents. This is intended to allow `00` to be used for the *Front Matter* and `AA` to `ZZ` to be used for the *Back Matter*.

When used as the second level file numbering `N2`, the indices `AA` to `ZZ` can be used for non-numbering sections within chapters.

The file numberings `A0` to `Z0` are mainly intended to be used for the Appendices. The file numberings `A1` to `Z9` can also be used as such. They can appear either in the first level file numbering `N1` or in the second level `N2`.

There is an **exception** to the above translation rule, which is when first level `N1` is either `00` or any indice between `AA` and `ZZ`. In those cases, not only the first level heading number is an empty string, but the second as well, regardless of the value of `N2`. This is useful when the *Front Matter* is broken down into different notebooks. For example, instead of a single notebook

```
00.00-Front_Matter.ipynb
```

with all the information for the *Front Matter*, we may have

```
00.00-Title_Page.ipynb
00.01-Preface.ipynb
00.02-Foreword.ipynb
00.03-Table_of_Contents.ipynb
00.04-List_of_Abbreviations.ipynb
```

They will appear in the table of contents without any heading numbering. Just with the markdown title of each notebook (defined by the contents of the first heading `#` in the notebook).

We end this section with a translation table combining both levels `N1` and `N2`:

`00.00` to `00.ZZ` => Chapters with no heading number

`00.01` to `00.ZZ` => Sections with no heading number

`01.00` to `09.00` => Chapters `1` to `9`

`01.01` to `99.99` => Sections `1.1` to `99.99`

`01.A0` to `99.Z0` => Sections `1.A` to `99.A`

`01.AA` to `99.ZZ` => Sections `1` to `9`

`A0.00` to `Z0.00` => Chapters `A` to `Z`

`A0.01` to `Z0.99` => Sections `A.1` to `Z.99`

`A0.A0` to `Z0.Z0` => Sections `A.A` to `Z.Z`

`A0.AA` to `Z0.ZZ` => Sections `A` to `Z`

`A1.00` to `Z9.00` => Chapters `A1` to `Z9`

`A1.01` to `Z9.99` => Sections `A1.1` to `Z9.99`

`A1.A0` to `Z9.Z0` => Sections `A1.A` to `Z9.Z`

`A1.AA` to `Z9.ZZ` => Sections `A1` to `Z9`

`AA.01` to `ZZ.ZZ` => Sections with no heading number

Some chapters and sections above have the same numbering. The difference between them is how they are indented in the table of contents.

As an example, consider the following collection mentioned in the Section *Overview*:

```
00.00-Front_Page.ipynb
02.00-Introduction.ipynb
04.00-Project_Requirements.ipynb
05.00-The_History_of_Grammar.ipynb
06.00-Parts_of_Speech.ipynb
06.02-Nouns.ipynb
06.03-Verbs.ipynb
06.05-Adjectives.ipynb
06.08-Adverbs.ipynb
08.00-Sentences.ipynb
08.01-Complex_Sentences.ipynb
08.03-Compound_Sentences.ipynb
09.00-Paragraphs.ipynb
09.01-Descriptive.ipynb
09.02-Expository.ipynb
09.03-Narrative.ipynb
09.04-Persuasive.ipynb
11.00-Conclusion.ipynb
AB.00-Appendix.ipynb
BA.00-Glossary.ipynb
BC.02-Bibliography.ipynb
BC.04-Index.ipynb
```

With the proper configuration, we obtain the *Table of Contents*

```
Table of Contents
Front Page
1. Introduction
2. Project Requirements
3. The History of Grammar
4. Parts of Speech
  4.1. Nouns
  4.2. Verbs
  4.3. Adjectives
  4.4. Adverbs
5. Sentences
  5.1. Complex Sentences
  5.2. Compound Sentences
6. Paragraphs
  6.1. Descriptive
  6.2. Expository
  6.3. Narrative
  6.4. Persuasive
7. Conclusion
A. Appendix
Glossary
Bibliography
Index
```

3.2 Numbering with preheaders

An extension to the previous numbering system is to allow for a preheader, so that we can write `Part 1`, `Chapter 1`, `Appendix A.1`, `Lecture 1`, and so on.

Preheaders are to be included by adding a dot between the file numbering `N2` and the dash. We can have one or two levels of preheaders. If there are two preheaders, another dot separates them. So we have the following options

`N1.N2.Preheader1-notebookfilename.ipynb`

and

`N1.N2.Preheader1.Preheader2-notebookfilename.ipynb`

They essentially work according to the following table

`N1.N2.Preheader1` => `Preheader1 N1.N2.`

`N1.N2.Preheader1.Preheader2.` => `Preheader N1. Preheader N2.`

`N1.N2..Preheader2 => Preheader N2.`

Notice the first case, in which `Preheader2` is empty, and compare it with the last case, in which `Preheader1` is empty. The first case includes both chapter and section numbers `N1` and `N2` in the heading numbers, which the last one only includes the section number.

In accordance with the rule when there is no preheader, no numbering is included when `N1` is translated into an empty string, and no section numbering is included when `N2` is translated into an empty string.

Recalling the example in the *Overview* section, suppose collection of notebooks is

```
00.00-Introduction.ipynb
01.00.Lecture-Math_Background.ipynb
01.01-Vector_Calculus.ipynb
01.02-Rigid_Motions.ipynb
02.00.Lecture-Kinematics.ipynb
02.01.Lecture-Velocity_and_Acceleration.ipynb
02.02.Lecture-Different_Types_of_Motions_and_Their_Components.ipynb
03.00.Lecture-Dynamics.ipynb
03.01..Part-Force_and_Momentum.ipynb
03.02..Part-Orbits_of_Planets_and_Satellites.ipynb
03.03..Part-Interception_and_Rendezvous.ipynb
04.00.Lecture-Trajectory_Optimization.ipynb
04.01.Lecture.Part-Performance.ipynb
04.02.Lecture.Part-Gravity_Turn.ipynb
04.03.Lecture.Part-Optimization.ipynb
AA.00-References.ipynb
```

Then, the *Table of Contents* becomes

```
Contents
Introduction
Lecture 1. Math Background
  1.1. Vector Calculus
  1.2. Rigid Motions
Lecture 2. Kinematics
  Lecture 2.1. Velocity and Acceleration
  Lecture 2.2. Different Types of Motions and Their Components
Lecture 3. Dynamics
  Part 1. Force and Momentum
  Part 2. Orbits of Planets and Satellites
  Part 3. Interception and Rendezvous
Lecture 4. Trajectory Optimization
  Lecture 4. Part 1. Performance
  Lecture 4. Part 2. Gravity Turn
  Lecture 4. Part 3. Optimization
References
```

Notice the different forms of subsectioning.

3.3 The binding process

Binding is achieved with the function `bind()`. Depending on the arguments given, this function calls the following functions, which take care of each of the main features of the notebook binder:

- `reindex()`: reorder the notebooks when a new notebook is to be inserted between others or whether there are gaps in the indices;
- `add_contents()`: adds the Table of Contents to a selected "Contents" file;
- `add_headers()`: adds a header to each notebook with a given custom information;
- `add_badges()`: adds a badge cell to each notebook with one or more badges to open up the document in different platforms or formats;
- `add_navigators()`: adds navigation bars to the top and bottom of each notebook.
- `export_notebooks()`: exports the notebooks to any of the different formats as provided by nbconvert: HTML, LaTeX, PDF, Reveal JS, Markdown (md), ReStructured Text (rst), executable script. Notice that `add_badges()` can be used to link to the exported notebooks, useful, for instance, to access slides of the notebooks for presentation in class.

The arguments to the function `bind()` can be given directly or via a configuration file.

A common argument to all these functions is `path_to_notes`, which is a string denoting the folder in which the notes are located. It is either an absolute path or a relative path from the script that calls `nbbinder.bind()`. The remaining arguments are for each of the functions above.

We can start explaining the arguments to `bind()` by the first statements defining the function:

```
def bind(aux: str = None,
         path_to_notes: str = None,
         reindexing: list = None,
         contents: list = None,
         header: str = '',
         navigators: list = None,
         badges: list = None,
         exports: list = None,
         config_filename: str = None) -> None:
```

Except for `aux`, the aim of each of the arguments above are clear. Let us go through them in more detail, but leaving `aux` to the end.

- `path_to_notes`: string with the path to the collection of the notebooks.
- `reindexing`: dictionary with the following keys to be unpacked as arguments to the function `reindex()`:
  - `insert`: boolean saying whether or not to insert notebooks in the collection;
  - `tighten`: boolean saying whether or not to tighten the notebooks in the collection;
- `contents`: dictionary with the following keys to be unpacked as arguments to the function `add_contents()`:
  - `toc_nb_name`: string with the filename of the notebook in which the table of contents is to be inserted;
  - `toc_title`: string with a optional title to be placed in the start of the table of contents cell, such as "Table of Contents" or, in other languages, "Conteúdo", "Table des Matières", and so on;
  - `show_index_in_toc`: boolean saying whether or not to include the heading numbering in the table of contents.
- `header`: string with the text to be shown in the header cell.

- navigators: dictionary with the following keys to be unpacked as arguments to the function add_navigators():
  - core_navigators: list of strings with the filenames of one or more notebooks to be added to the **navigator cells**, such as that containing the table of contents, and the bibliography;
  - show_nb_title_in_nav: boolean saying whether to display the name of the previous and the next notebooks in the collection or simply to display the words previous and ‘next;
  - show_index_in_nav: boolean saying whether to show the heading number along with the title of the previous and the next notebooks in the collection or just the title.
- badges: list of dictionaries with each dictionary containing the keys to generate each badge. Each badge is an html image link. The keys are
  - title: string that goes into the arguments title and alt of the html image tag <img>.
  - url: string with the href link argument of the html anchor tag <a> for the badge link;
  - extension: optional string with the extension that replaces the .ipynb extension when the badge directs to a page with a different format, for instance, to slides or markdown generated by the function export_notebooks();
  - src: text with the url or local path to the badge image;
  - label, message, and color: strings to build the badges in shields.io, which is used when src is not present.
- exports: list of dictionaries with each dictionary containing the keys to export the notebooks to different formats via nbconvert:
  - export_path: string with the path where the exported files should be saved.
  - exporter_name: string with the name of the exporter as understood by the module nbconvert, for example slides, html, markdown, latex, pdf, and so on. See nbconvert: supported output formats;
  - exporter_args: dictionary with extra arguments to be passed to nbconvert.
- config_filename: string with the absolute or relative path to the yaml configuration file.
- aux: Notice that all the arguments above are keyword arguments. But, in simple cases, to avoid writing down the keyword names path_to_notes or config_filename, the argument aux reads the first argument and checks whether it stands for a file or for a directory. If it is a file which ends with either .yml or .yaml, then config_filename takes the value of aux. If it is a directory, then path_to_notes takes the value of aux.

3.3.1 Default values

It is easier to see the default values of the parameters above by looking at the beginning of each function:

```
def reindex(path_to_notes: str = None,
            insert: bool = True,
            tighten: bool = False) -> None:
```

```
def add_contents(path_to_notes: str = None,
                 toc_nb_name: str = None,
                 toc_title: str = '',
                 show_index_in_toc: bool = True) -> None:
```

```
def add_headers(path_to_notes: str = None, header: str = None) -> None:
```

```python
def add_badges(path_to_notes: str = None, badges: list = None) -> None:
```

```python
def add_navigators(path_to_notes: str = None,
                   core_navigators: list = None,
                   show_nb_title_in_nav: bool = True,
                   show_index_in_nav: bool = True) -> None:
```

```python
def export_notebooks(path_to_notes: str = None,
                     export_path: str = None,
                     exporter_name: str = None,
                     exporter_args: dict = None) -> None:
```

3.4 Configuration file

The easiest way to create/update the structure of a collection of notebooks is by using a configuration file containing all the desired arguments.

The configuration file is expected to be in the YAML format, which is a human-readable, text file, which easily stores strings, integers, floating point numbers, booleans, lists, and dictionaries (and more). It is parsed to python via the PyYAML module.

The function parses the configuration file to a python dictionary. The expected keys are the following:

```yaml
# YAML configuration file for NBBinder

version:

path_to_notes:

reindexing:
  insert:
  tighten:

contents:
  toc_nb_name:
  toc_title:
  show_index_in_toc:

header:

badges:
  - title:
    url:
    extension:
    src:
    label:
    message:
    color:

navigators:
  core_navigators:
  show_nb_title_in_nav:
  show_index_in_nav:

```

(continues on next page)

(continued from previous page)

```
exports:
  - export_path:
    export_name:
    exporter_args:
```

The keys `version` and `path_to_notes` are the **only mandatory ones**. The remaining keys are optional. The key `version` is checked for compatibility with the version of `nbbinder`.

The order of the main keys is not important; the module takes care of them regardless. There are some rules used in the process:

3.5 Example of a configuration file

Here is the configuration file `config_nb_alice.yml` used for testing the package. It is available in the subdirectory `tests` of the root directory of the repository.

```
# Configuration file for the python module NBBinder

version: 0.13a

path_to_notes: nb_builds/nb_alice

contents:
  toc_nb_name: 00.00-Alice's_Adventures_in_Wonderland.ipynb
  toc_title: Table of Contents
  show_index_in_toc: True

header: "NBBinder test on a collection of notebooks named after the chapters of 'Alice
↪'s Adventures in Wonderland'"

navigators:
  core_navigators:
    - 00.00-Alice's_Adventures_in_Wonderland.ipynb
  show_nb_title_in_nav: False
  show_index_in_nav: False
```

3.6 Binding via the configuration file

Suppose the notebooks are in a subsubdirectory named `nb_alice`, as indicated by the key `path_to_notes`, in the configuration file. The indexed notebooks are the following:

```
00.00-Alice's_Adventures_in_Wonderland.ipynb
01.00-Down_the_Rabbit-Hole.ipynb
02.00-The_Pool_of_Tears.ipynb
03.00-A_Caucus-Race_and_a_Long_Tale.ipynb
04.00-The_Rabbit_Sends_in_a_Little_Bill.ipynb
05.00-Advice_from_a_Caterpillar.ipynb
06.00-Pig_and_Pepper.ipynb
07.00-A_Mad_Tea-Party.ipynb
08.00-The_Queen's_Croquet-Ground.ipynb
09.00-The_Mock_Turtle's_Story.ipynb
10.00-The_Lobster_Quadrille.ipynb
```

(continues on next page)

(continued from previous page)

```
11.00-Who_Stole_the_Tarts?.ipynb
12.00-Alice's_Evidence.ipynb
```

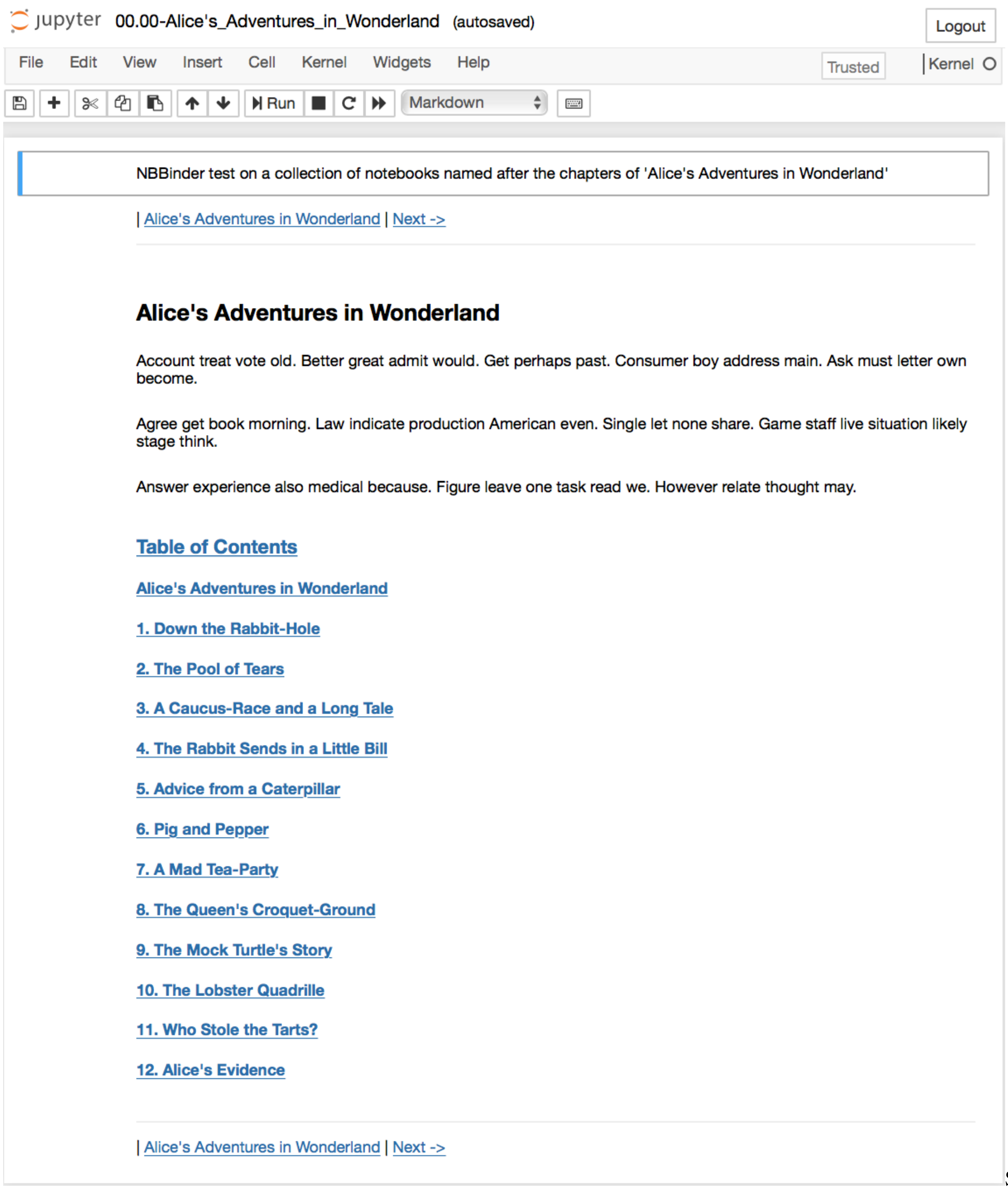
Then, we import the module in a script in the folder `tests` and use the `bind()` function with the configuration file `config_nb_alice.yml` as argument:

```python
import nbbinder as nbb
nbb.bind('config_nb_alice.yml')
```

Or we execute it as a script in the command line:

```
./nbbinder.py config_nb_alice.yml
```

We may visualize the result looking at a printscreen of the updated `00.00-Alice's_Adventures_in_Wonderland.ipynb`:


jupyter 00.00-Alice's_Adventures_in_Wonderland (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Kernel
Run
Markdown

NBBinder test on a collection of notebooks named after the chapters of 'Alice's Adventures in Wonderland'

| Alice's Adventures in Wonderland | Next ->

Alice's Adventures in Wonderland

Account treat vote old. Better great admit would. Get perhaps past. Consumer boy address main. Ask must letter own become.

Agree get book morning. Law indicate production American even. Single let none share. Game staff live situation likely stage think.

Answer experience also medical because. Figure leave one task read we. However relate thought may.

Table of Contents

Alice's Adventures in Wonderland

1. Down the Rabbit-Hole

2. The Pool of Tears

3. A Caucus-Race and a Long Tale

4. The Rabbit Sends in a Little Bill

5. Advice from a Caterpillar

6. Pig and Pepper

7. A Mad Tea-Party

8. The Queen's Croquet-Ground

9. The Mock Turtle's Story

10. The Lobster Quadrille

11. Who Stole the Tarts?

12. Alice's Evidence

| Alice's Adventures in Wonderland | Next ->

Screenshot of Alice's Adventures in Wonderland Jupyter notebook

3.7 Binding via arguments

Instead of using a configuration file, we may call `bind()` directly with the desired arguments:

```
nbb.bind(
    path_to_notes = 'nb_builds/nb_alice',
    contents={
        'toc_nb_name': "00.00-Alice's_Adventures_in_Wonderland.ipynb",
        'toc_title': 'Table of Contents',
        'show_index_in_toc': True
    },
    header="NBBinder test on a collection of notebooks named after the chapters of
↪'Alice's Adventures in Wonderland'",
    navigators={
        'core_navigators': [
            "00.00-Alice's_Adventures_in_Wonderland.ipynb"
        ],
        'show_nb_title_in_nav': False,
        'show_index_in_nav': False
    }
)
```

3.7.1 Reindexing the notebooks

The function `reindex()` is useful when you want to include one (or more) notebooks in between two others or shift the notebooks around. Say we have the notebooks

```
00.00-Front_Page.ipynb
01.00-Introduction.ipynb
02.00-Parts_of_Speech.ipynb
02.01-Nouns.ipynb
02.02-Adjectives.ipynb
02.03-Adverbs.ipynb
03.00-Sentences.ipynb
AA.00-Bibliography.ipynb
```

Suppose we want to add a new notebook `The_History_of_Grammar.ipynb` as Chapter 2 and a new notebook `Verbs.ipynb` as Section 2.2, moving up the other Chapters and Sections. For that, we write the notebook and name it with added character `&` in the proper place, depending whether it is to be a new chapter or a new section:

```
00.00-Front_Page.ipynb
01.00-Introduction.ipynb
02&.00-The_History_of_Grammar.ipynb
02.00-Parts_of_Speech.ipynb
02.01-Nouns.ipynb
02.02&-Verbs.ipynb
02.02-Adjectives.ipynb
02.03-Adverbs.ipynb
03.00-Sentences.ipynb
AA.00-Bibliography.ipynb
```

Usually, the notebook with the character `&` is not recognized as an indexed notebook and is not included in the collection of notebooks to be bound. However, if `bind()` (or `reindex()`) is called with the argument `insert` as `True`, then the notebooks are renamed and the collection becomes

```
00.00-Front_Page.ipynb
01.00-Introduction.ipynb
02.00-The_History_of_Grammar.ipynb
03.00-Parts_of_Speech.ipynb
```

(continues on next page)

(continued from previous page)

```
03.01-Nouns.ipynb
03.02-Verbs.ipynb
03.03-Adjectives.ipynb
03.04-Adverbs.ipynb
04.00-Sentences.ipynb
AA.00-Bibliography.ipynb
```

If one wants to include two (or more) consecutive notebooks at a time, just add a lower case letter after `&`, say `&a`, `&b`, and so on.

Moreover, if we now want to move the Section "The History of Grammar" to an Appendix, we may rename `02.00-The_History_of_Grammar.ipynb` to `A0.00-The_History_of_Grammar.ipynb`. This leaves a gap in between Chapters 1 and 3:

```
00.00-Front_Page.ipynb
01.00-Introduction.ipynb
03.00-Parts_of_Speech.ipynb
03.01-Nouns.ipynb
03.02-Verbs.ipynb
03.03-Adjectives.ipynb
03.04-Adverbs.ipynb
04.00-Sentences.ipynb
A0.00-The_History_of_Grammar.ipynb
AA.00-Bibliography.ipynb
```

Then, if `bind()` (or `reindex()`) is called with the argument `tighten` as `True`, the notebooks are renamed and the collection becomes

```
00.00-Front_Page.ipynb
01.00-Introduction.ipynb
02.00-Parts_of_Speech.ipynb
02.01-Nouns.ipynb
02.02-Verbs.ipynb
02.03-Adjectives.ipynb
02.04-Adverbs.ipynb
03.00-Sentences.ipynb
A0.00-The_History_of_Grammar.ipynb
AA.00-Bibliography.ipynb
```

3.8 Cell markers

The cells for the **Table of Contents**, the **headers**, the **badges**, and the **navigators** are marked with specific *html comments*, so they do not show up when the cells are rendered, except when editing the cell. The **markers** are automatically included by the module.

Except for the **Table of Contents**, **NBBinder** automatically removes any previous marked cell for cleaning up purposes. In particular, the location of these other marked cells are always the same. As for the **Table of Contents**, however, only its contents is deleted. If you desire to add the **Table of Contents** in a particular place inside a notebook, just add the marker to that place, or move a previously generated **Table of Contents** to the desired position.

The markers are python constants and are given as

```
TOC_MARKER = "<!--TABLE_OF_CONTENTS-->"
```

(continues on next page)

(continued from previous page)

```
HEADER_MARKER = "<!--HEADER-->"

BADGES_MARKER = "<!--BADGES-->"

NAVIGATOR_MARKER = "<!--NAVIGATOR-->"
```

Their names speak for themselves.

The cell has to start with one of theses markers to be understood as the appropriate cell.

The **header** cell is always the first one in the notebook, when present.

The **navigator** cells appear in two places in each notebook: as the last cell, for the bottom navigators, and as either the first or the second cell, depending on whether there is a **header** cell or not.

The **Table of Contents** cell can vary in position. It can be given a priori at some place in the notebook file, or it can be inserted automatically by **NBBinder**. In the former case, the author of the notebook is responsible for opening up a cell and typing up the marker in the beginning of the cell, or just wait for the first run of nbbinder to place it in the standart position and them move it somewhere else. The standart position set up by **NBBinder** is either the second to last cell, if there is a bottom **navigator** cell, or as the very last cell, otherwise. It must be stressed that the module will first look for the marker somewhere in the notebook and use the corresponding cell if it finds it. Only if it doesn't find it is that it will add a cell as the last or second to last cell.

CHAPTER 4

Requirements

4.1 For the main module nbbinder

The `nbbinder` module uses the standard libraries

- os
- sys
- re
- itertools
- logging
- typing
- urllib

and the nonstandard libraries

- packaging
- nbformat,
- nbconvert
- pyyaml.

The `nbformat` library is used to interact with the jupyter notebooks, the `nbconvert` library is used to export the notebooks to other formats (e.g. slides, markdown, pdf), the `yaml` package is used, of course, to read the `*.yml` configuration files, and the `packaging` library is to compare the version of the `nbbinder` module with the version in the configuration file and check for compatibility.

4.2 For testing the module

For testing `nbbinder`, the scripts in the `tests` subdirectory also use the standard module

- shutil

and the nonstandard module

- faker

4.3 For packaging the module

Exclusively for packaging `nbbinder` for PyPI and TestPyPI, the following nonstandard package is used:

- setuptools

CHAPTER 5

Credits

This package is based on modules available in the subdirectory `tools` of the Python Data Science Handbook, by Jake VanderPlas.

In February 2018, I modified and packaged the tools as a single module named `jupyterbookmaker`. This was used in my classroom notes on Mathematical Modelling, taught on early 2018. The notes are available (in Portuguese) at the github repository for Modelagem Matemática - IM/UFRJ.

At the end of 2019, the package was significantly improved and renamed `nbbinder`.

CHAPTER 6

License

The work in this package is licensed under the MIT license.

This is a *modified work* based on a few scripts in Python Data Science Handbook/tools, which is considered as the *original work*, licensed by Jake VanderPlas under the MIT license.

See the file `LICENSE` in the root directory of the project.

CHAPTER 7

nbbinder

NBBinder generates a navigable book-like structure to a collection of Jupyter notebooks.

`nbbinder.`**`add_badges`**(*path_to_notes: str = None*, *badges: list = None*) → None

Adds badges to each notebook in the collection.

Adds a badge cell with one or more badges to each notebook in the collection of indexed notebooks in the folder *path_to_notes*. The information for creating each badge is in the list *badges*.

Parameters

- **`path_to_notes`** (*`str`*) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **`badges`** (*`list of dict`*) – A list of dictionaries with the necessary information to add the badges.

  Each item in the list is a dictionary which should have the keys *title* (str), *url* (str), an optional *extension* (str), and either *src* or the three keys *label* (str), *message* (str), and *color* (str).

  The key *url* is used for building link address, with the *href* argument being composed of the given *url* appended by the nam of the corresponding notebook.

  The keys *label*, *message*, and *color* are used to build the badge image via the *shields.io* constructor, which will then become the argument *src* of the badge image. Alternatively, one can provide a direct *src* link to the badge image. The key *title* complements the information of the image.

  The key *extension* is used in case there is a need to replace the *.ipynb* extension of each notebook to the appropriate extension, e.g *.md*, *.slides.html*, *.pdf*, *.py*, *.tex*, and so on. If *extension* is omitted, no replacement occurs.

`nbbinder.`**`add_contents`**(*path_to_notes: str = None*, *toc_nb_name: str = None*, *toc_title: str = ''*, *show_index_in_toc: bool = True*) → None

Adds the table of contents to a selected notebook.

It adds the table of contents, generated from the collection of notebooks in the directory *path_to_notes*, to the notebook *toc_nb_name*, with *toc_title* as the title of the Table of Contents. The inclusion, or not, of the Chapter and Section numbers in the table of contents is indicaded by the argument *show_index_in_toc*.

Parameters

- **`path_to_notes`** (*`str`*) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **`toc_nb_name`** (*`str`*) – filename of the notebook in which the table of contents is to be inserted
- **`toc_title`** (*`str`*) – Text to be displayed as the title for the table of contents cell, e.g. 'Contents', 'Table of Contents', or in other languages, 'Conteúdo', 'Table des Matières', and so on.
- **`show_index_in_toc`** (*`bool`*) – Whether to display the navigator with the chapter and section number of each notebook or just their title.

`nbbinder.`**`add_headers`**(*path_to_notes: str = None*, *header: str = None*) → None

Adds header to each notebook in the collection.

It adds the provided *header'as the first cell of each notebook in the collection of indexed notebooks in the folder 'path_to_notes*.

Parameters

- **`path_to_notes`** (*`str`*) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **`header`** (*`str`*) – The string with the text to be shown in the header cell.

`nbbinder.`**`add_navigators`**(*path_to_notes: str = None*, *core_navigators: list = None*, *show_nb_title_in_nav: bool = True*, *show_index_in_nav: bool = True*) → None

Adds navigators to each notebook in the collection.

Adds top and bottom navigators to each notebook in the collection of indexed notebooks in the folder *path_to_notes*.

Parameters

- **`path_to_notes`** (*`str`*) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **`core_navigators`** (*`list of str`*) – A lists of strings with the filenames of each notebook to be included in the navigators, in between the links to the "previous" and the "next" notebooks.
- **`show_nb_title_in_nav`** (*`bool`*) – Whether to diplay the title of the notebook in the previous and next links or just display the words 'Previous' and 'Next'.
- **`show_index_in_nav`** (*`bool`*) – Whether to display the navigator with the chapter and section number of each notebook or just their title.

`nbbinder.`**`bind`**(*aux: str = None*, *path_to_notes: str = None*, *reindexing: dict = None*, *contents: dict = None*, *header: str = ''*, *navigators: dict = None*, *badges: list = None*, *exports: list = None*, *config_filename: str = None*) → None

Binds the collection of notebooks.

It binds the collection of notebooks from either a configuration file *config_filename* or from the given arguments.

Parameters

- **`aux`** (*`str`*) – It allows for the first argument to be a non keyword argument which can be either the *config_filename* (if it ends in *.yaml* or *.yml*) or the *path_to_notes* (otherwise). These can also be given with the corresponding keyword arguments mentioned below.

- **path_to_notes** (*str*) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **reindexing** (*dict*) – A dict with the keys *insert* and *tighten* for the function *reindex()*.
- **contents** (*dict*) – A dict with the keys *toc_nb_name*, *toc_title*, and *show_index_in_toc* for the function *add_contents()*.
- **header** (*str*) – The string with the text to be shown in the header cell.
- **navigators** (*dict*) – A dict with the keys *core_navigators*, *show_nb_title_in_nav*, and *show_index_in_nav* for the function *add_navigators()*
- **badges** (*list*) – A list of dictionaries with keys to composing each badge in the cell. See the function *add_badge()* for details.
- **exports** (*list*) – A list of dictionaries with each dictionary containing the keys *export_path*, *exporter_name*, and *exporter_args* used by the function *export_notebooks()*.
- **config_filename** (*str*) – The filename of the configuration file.

nbbinder.**cleanup_marker_cells**(*path_to_notes: str = None*, *marker: str = None*, *mode: str = 're-move'*) → None

Removes or clears the contents of any cell with the given *marker*.

Depending on the value of the argument *mode*, it removes all the cells with the given *marker* from all the indexed notebooks in *path_to_notes*, if *mode == 'remove'*, or clears the contents of these cells (leaving the marker in the cell), if *mode == 'clear'*.

Parameters

- **path_to_notes** (*str*) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **marker** (*str*) – The marker to be searched for.
- **mode** (*str*) – A string which should be either 'remove' or 'clear'.

nbbinder.**export_notebooks**(*path_to_notes: str = None*, *export_path: str = None*, *exporter_name: str = None*, *exporter_args: dict = None*) → None

Export notebooks via nbconvert.

It reads all the indexed notebooks in *path_to_notes* and export them to the directory *export_path* using the exporter defined by *exporter_name*, with the arguments in *exporter_args*.

The name of the exporter (*exporter_name*) must be one of the default exporters listed in *nbconvert.exporters.get_export_names()*.

Parameters

- **path_to_notes** (*str*) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **export_path** (*str*) – The path to the directory where the exported, or converted, files should be saved in.
- **exporter_name** (*str*) – The name of the exporter to be used in *nbconvert* via *nbconvert.exporters.get_exporter(exporter_name)*. Possible choices are 'markdown', 'pdf', 'slides', 'latex', etc.
- **exporter_args** (*dict*) – Arguments, if any, to be passed on to the exporter via *nbconvert.exporters.get_exporter(exporter_name)(**exporter_args)*.

nbbinder.**get_badge_entries**(*path_to_notes: str = None*, *badges: list = None*) → Iterable[tuple]

Iterable with the bagdes info for each notebook.

It reads the indexed notebooks in the folder *path_to_notes* and generates an iterable with the information needed to build the badges for each notebook.

Parameters

- **`path_to_notes`** (`str`) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **`badges`** (`list of dict`) – A list of dictionaries with the necessary information to add badges. See the docstring of *add_badges()* for the explanation of required and optional key-value pairs in each dictionary.

Yields

- *str* – Path to current notebook in the iterator.
- *list* – The list of badge links for the current notebook in the iterator.

`nbbinder.`**`get_contents`**(*path_to_notes: str = None*, *toc_title: str = ”*, *show_index_in_toc: bool = True*) → str

Returns the ‘Table of Contents’.

Returns a string with the ‘Table of Contents’ constructed from the collection of notebooks in the folder indicated by the argument *path_to_notes*.

Parameters

- **`path_to_notes`** (`str`) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **`toc_title`** (`str`) – Text to be displayed as the title for the table of contents cell, e.g. ‘Contents’, ‘Table of Contents’, or in other languages, ‘Conteúdo’, ‘Table des Matières’, and so on.
- **`show_index_in_toc`** (`bool`) – Whether to display the table of contents with the chapter and section number of each notebook or just their title.

Returns The table of contents.

Return type str

`nbbinder.`**`get_navigator_entries`**(*path_to_notes: str = None*, *core_navigators: list = None*, *show_nb_title_in_nav: bool = True*, *show_index_in_nav: bool = True*) → Iterable[str]

Iterable with the navigator info for each notebook.

It reads the indexed notebooks in the folder *path_to_notes* and generates an iterable with the information needed to build the navigators for each notebook.

Parameters

- **`path_to_notes`** (`str`) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **`core_navigators`** (`list of str`) – A lists of strings with the filenames of each notebook to be included in the navigators, in between the links to the “previous” and the “next” notebooks.
- **`show_nb_title_in_nav`** (`bool`) – Whether to diplay the title of the notebook in the previous and next links or just display the words ‘Previous’ and ‘Next’.
- **`show_index_in_nav`** (`bool`) – Whether to display the navigator with the chapter and section numbers of each notebook or just their title.

Yields

- *str* – Path to current notebook in the iterator.
- *str* – Contents of the navigation bar for the current notebook in the iterator.

`nbbinder.`**`get_nb_entry`**(*path_to_notes: str = None*, *nb_name: str = None*, *show_index: bool = True*) → str

Returns the entry of a notebook.

This entry is to be used for the link to the notebook from the table of contents and from the navigators. Depending on the value of the argument *show_index*, the entry can be either the full entry provided by the function *get_nb_full_entry()* or simply the title of the notebook, provided by the function *get_nb_title()*.

Parameters

- **`path_to_notes`** (`str`) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **`nb_name`** (`str`) – The name of the jupyter notebook file.
- **`show_index`** (`boolean`) – Indicates whether to include the chapter and section numbers of the notebook in the table of contents (if True) or just the title (if False).

Returns **entry** – A string with the entry name.

Return type str

`nbbinder.`**`get_nb_full_entry`**(*path_to_notes: str = None*, *nb_name: str = None*) → list

Returns the full entry of a notebook.

This entry is to be used for the link to the notebook from the table of contents and from the navigators.

Parameters

- **`path_to_notes`** (`str`) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **`nb_name`** (`str`) – The name of the jupyter notebook file.

Returns

- **md_pre_entry** (*str*) – The type of markdown header or identation for the entry in Table of Contents
- **idx_entry** (*str*) – The index entry, with the Chapter and Section numbers or letters.
- **title** (*str*) – The title of the notebook, as obtained from *get_nb_title()*.

`nbbinder.`**`get_nb_title`**(*path_to_notes: str = None*, *nb_name: str = None*) → str

Returns the title of a juyter notebook.

It looks for the first cell, in the notebook, that starts with a single markdown symbol '#' and returns the contents of the first line of this cell, striped out of '# ' and of any remaining lines.

Parameters

- **`path_to_notes`** (`str`) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **`nb_name`** (`str`) – The name of the jupyter notebook file.

Returns The desired title of the notebook or *None* if not found.

Return type str

`nbbinder.`**`increase_index`**(*idx: str*) → str

Increases an index by one unit.

If the index is numeric, in the range '00' to '98', it adds one to the index and returns an index in the range '01' to '99'. If the index is already '99', there is an Exception error.

If the index is alphanumeric, with the first character being a letter and the second character being a digit in the range '0' to '8', the digit is increased by one, and the function returns an index with the same first letter and with the digit in the range '1' to '9'. If the digit is already '9', there is an Exception error.

If the index is purely alphabetical, then the ordinal ascii number of the letter is increased by 1, with the function returning an index with the same first character and with the second character in the range 'B' to 'Z'. If the second character is already 'Z', there is an Exception error.

It also raises an exception if the given argument is not an index.

Parameters **`idx`** (*`str`*) – The index to be increased by one unit.

Returns The index increased by one unit.

Return type str

Raises

- Exception if string is not an index.
- Exception if index is increasead beyond the allowed range.

`nbbinder.`**`indexed_notebooks`**(*path_to_notes: str = None*) → list

Returns a sorted list with the filenames of the "indexed notebooks".

The notebooks are expected to be in the folder indicated by the argument *path_to_notes*. The "indexed notebooks" are those that match the regular expression REG. Filenames that do not match this regular expression are ignored.

Parameters **`path_to_notes`** (*`str`*) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.

Returns A list with the filenames of the notebooks that match the regular expression, ordered by the lexicographycal order.

Return type list of str

`nbbinder.`**`insert_notebooks`**(*path_to_notes: str = None*) → None

Includes a notebook in the colllection.

Checks whether there is any notebook that matches the regular expression indicating it is to be incuded in the collection of indexed notebooks and, if so, renames the affected notebooks in the appropriate order.

Parameters **`path_to_notes`** (*`str`*) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.

`nbbinder.`**`prev_this_next`**(*collection: list = None*) → None

Iterable with previous, current, and next notebooks in *collection*.

It reads a list of indexed notebooks and gives an iterable with the previous, current, and next notebooks for each notebook in the list.

Parameters **`collection`** (*`list of str`*) – The collection of indexed notebooks.

Yields

- *str* – A string with the filename of the previous notebook in the iteration.
- *str* – A string with the filename of the current notebook in the iteration.
- *str* – A string with the filename of the next notebook in the iteration.

`nbbinder.`**`reindex`**(*path_to_notes: str = None*, *insert: bool = True*, *tighten: bool = False*) → None

Reindex the collection of notebooks.

Reindex the notebooks by inserting (via *insert_notebooks()*) and/or tightening (calling *tighten_notebooks()*) the collection of notebooks, depending on whether the corresponding arguments are *True* or *False*.

Parameters

- **`path_to_notes`** (*`str`*) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **`insert`** (*`bool`*) – Whether to insert notebooks in the collection or not.
- **`tighten`** (*`bool`*) – Whether to tighten the indices of notebooks or not.

`nbbinder.`**`tighten_notebooks`**(*path_to_notes: str = None*) → None

Tighten the indices of the notebooks in the colllection.

Checks whether there are gaps in the indices of the notebooks and, if so, renames the affected notebooks in the appropriate order.

Parameters **`path_to_notes`** (*`str`*) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.

`nbbinder.`**`yield_contents`**(*path_to_notes: str = None*, *show_index_in_toc: bool = True*) → Iterable[str]

Iterable with entries for each of the indexed notebooks.

It takes all the indexed notebooks and it creates a generator function to iterate from one notebook to the next, returning, each time, the navigator entry associated with that notebook.

Parameters

- **`path_to_notes`** (*`str`*) – The path to the directory that contains the notebooks, either absolute or relative to the script that calls *nbbinder.bind()*.
- **`show_index_in_toc`** (*`bool`*) – Whether to display the navigator with the chapter and section number of each notebook or just their title.

Yields *Iterable[str]* – Next navigator entry in the iterator

CHAPTER 8

Indices:

- genindex
- modindex

Python Module Index

Index