

# Sphinx-doc

Par GALODE Alexandre  

Date de publication : 26 septembre 2014

Dernière mise à jour : 26 septembre 2014

TOUT PUBLIC

Logiciel relativement jeune, Sphinx-doc a su s'imposer très rapidement dans le milieu professionnel comme l'outil indispensable pour générer de la documentation de qualité.

Ce merveilleux outil est libre, puissant et disposant d'une large communauté d'utilisateurs, je vous convie ici à le découvrir.

**Commentez.**

I - Introduction.....	4
II - Présentation.....	4
II-A - Historique.....	4
II-B - Template.....	4
II-B-1 - Les templates standards.....	4
II-B-2 - Personnaliser le template de base : default.....	5
II-B-3 - Quelques templates supplémentaires.....	6
II-C - Format de sortie.....	7
III - Structure.....	8
III-A - <code>_static</code> .....	8
III-B - <code>_templates</code> .....	8
III-C - <code>index.rst</code> .....	8
III-D - <code>conf.py</code> .....	8
III-D-1 - Langue.....	9
III-D-2 - Configuration HTML.....	9
IV - Le langage ReST.....	9
IV-A - historique.....	9
IV-B - Principe.....	9
IV-C - Fonctionnement.....	10
IV-C-1 - Les directives.....	10
IV-C-2 - Le fichier <code>index.rst</code> .....	10
IV-C-3 - Espaces et sauts de ligne.....	11
IV-D - Mise en forme du texte.....	11
IV-D-1 - Gras.....	11
IV-D-2 - Italique.....	11
IV-E - Titre.....	12
IV-F - Échappement.....	13
IV-G - Commentaires.....	13
IV-H - Liste.....	13
IV-H-1 - Classique.....	13
IV-H-2 - Numérotée.....	13
IV-I - Tableaux.....	14
IV-I-1 - Méthode 1.....	14
IV-I-2 - Méthode 2.....	14
IV-I-3 - Méthode 3.....	14
IV-J - Insertion de bloc.....	15
IV-J-1 - Texte/code depuis un fichier.....	15
IV-J-2 - Code directement saisi.....	15
IV-K - Lien Hypertexte et de téléchargement.....	17
IV-L - Image.....	17
IV-M - Boîte.....	17
IV-M-1 - Voir aussi.....	17
IV-M-2 - Note.....	17
IV-M-3 - Note de côté.....	18
IV-M-4 - Attention.....	18
IV-M-5 - A faire.....	18
IV-M-6 - Personnaliser.....	18
IV-N - Substitution.....	18
IV-O - Note de bas de page.....	19
IV-P - Citation.....	19
IV-Q - Expression mathématique avec <code>pngmath</code> .....	19
IV-R - Documentation Python manuelle.....	20
IV-S - Documentation Python automatique.....	21
IV-S-1 - Indiquer à Sphinx-doc où trouver le code Python.....	21
IV-S-2 - Côté fichiers <code>.rst</code> .....	21
IV-S-3 - Côté docstrings.....	22
V - Pratique : créer une simple documentation.....	23
V-A - Installation de Sphinx-doc.....	23

V-B - Création d'un nouveau projet.....	23
V-B-1 - Notre projet de démo.....	25
V-C - Génération de notre documentation au format HTML.....	26
V-D - Rendu final.....	27
VI - Pratique : autodocumenter son code Python.....	28
VI-A - Génération de notre documentation au format HTML.....	31
VI-B - Rendu final.....	32
VII - Conclusion.....	32
VIII - Liens.....	33
IX - Remerciements.....	33

## I - Introduction

Servant à la génération de documentation, Sphinx-doc est un logiciel libre qui, malgré sa jeunesse, est largement reconnu et utilisé dans le milieu professionnel.

Se reposant sur le langage Restructured Text, ou ReST, il permet à partir d'une même source de générer divers formats : HTML, LaTeX, man...

À travers cet article, nous allons voir l'ensemble des bases utiles et nécessaires permettant d'utiliser cet outil afin de créer de la documentation HTML et d'autodocumenter du code source Python.

## II - Présentation

### II-A - Historique

Créé en 2008 par Geord Brandl, Sphinx-doc visait alors à corriger un manque : la création de **documentation pour Python**.

Il a été peu à peu adopté pour d'autres usages par la suite, dont entre autres la création de documentation de code source.

À ce titre, il a alors officiellement remplacé des outils tels qu'Epydoc.

### II-B - Template

L'une des particularités, et des forces, de Sphinx-doc, c'est la possibilité de choisir un template permettant à la fois de modifier l'organisation de la page, mais également l'apparence visuelle (couleur).

Sphinx-doc est livré avec plusieurs templates de base, dont le template par défaut qui est extrêmement personnalisable.

Mais outre ces templates, vous pouvez également en trouver de nombreux sur le net, généreusement proposés par des membres de la communauté, avec plus ou moins d'options.

Enfin, sachez que si vous désirez développer votre propre template pour des raisons marketing par exemple, la **documentation officielle** vous fournit tous les éléments nécessaires à sa création.



*Pour avoir une idée pratique de l'application de template, je vous invite à visiter la documentation de **Python2.7** & **Python 3.X**.*



*Quasiment tous les templates offrent un lien surnommé « show source ». Ce lien permet d'afficher la source ReST de la page que vous visualisez. Ainsi, si un élément vous plaît, vous pouvez voir comment il a été codé.*

#### II-B-1 - Les templates standards

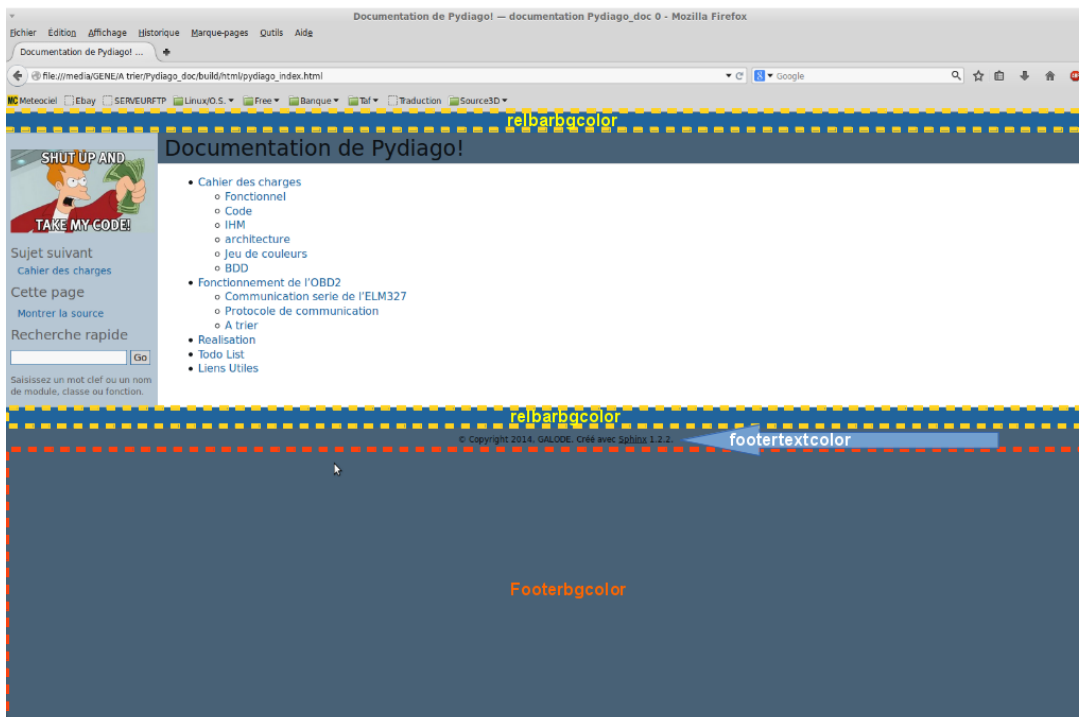
L'ensemble des thèmes livrés par défaut sont disponibles **ICI**.

Comme vous pourrez le constater, il est fort probable que vous ayez déjà croisé un site exploitant Sphinx-doc, par exemple la documentation officielle de Python.

## II-B-2 - Personnaliser le template de base : default

Nous allons ici nous attarder un peu sur le template « default ». Pourquoi ce thème ? Tout simplement parce qu'outre le fait d'être le thème par défaut, et donc être disponible de base, c'est également un des thèmes les plus personnalisables.

Voyons maintenant plus amplement comment le configurer :



Option	Type	Description
rightsidebar	true or false	Position de la barre de menu (à gauche (« false ») ou à droite).
stickysidebar	true or false	Barre de menu fixe (« false ») ou non.
externalrefs	true or false	Afficher d'une couleur différente les liens internes et externes.
footerbgcolor	CSS color	Couleur de fond du pied de page.
footertextcolor	CSS color	Couleur du texte du pied de page.
sidebarbgcolor	CSS color	Couleur de fond de la barre de menu.
sidebartextcolor	CSS color	Couleur du texte de titre dans la barre de menu.
sidebarlinkcolor	CSS color	Couleur de texte liens vers les pages de la doc.
relbarbgcolor	CSS color	Couleur des barres haut et bas.
relbartextcolor	CSS color	Couleur du texte de titre.
relbarlinkcolor	CSS color	Couleur du texte lien.
bgcolor	CSS color	Fond de la page normal.
textcolor	CSS color	Couleur du texte de la page.
linkcolor	CSS color	Couleur du lien non visité.
visitedlinkcolor	CSS color	Couleur du lien visité.
headbgcolor	CSS color	Couleur de fond des titres.
headtextcolor	CSS color	Couleur du texte de titre.
headlinkcolor	CSS color	Couleur des liens de titre
codebgcolor	CSS color	Couleur de fond pour le code.
codetextcolor	CSS color	Couleur de texte pour le code.
bodyfont	CSS font-family	Police pour le texte.
headfont	CSS font-family	Police pour les titres.



Les logos ne peuvent excéder 200 pixels de large.

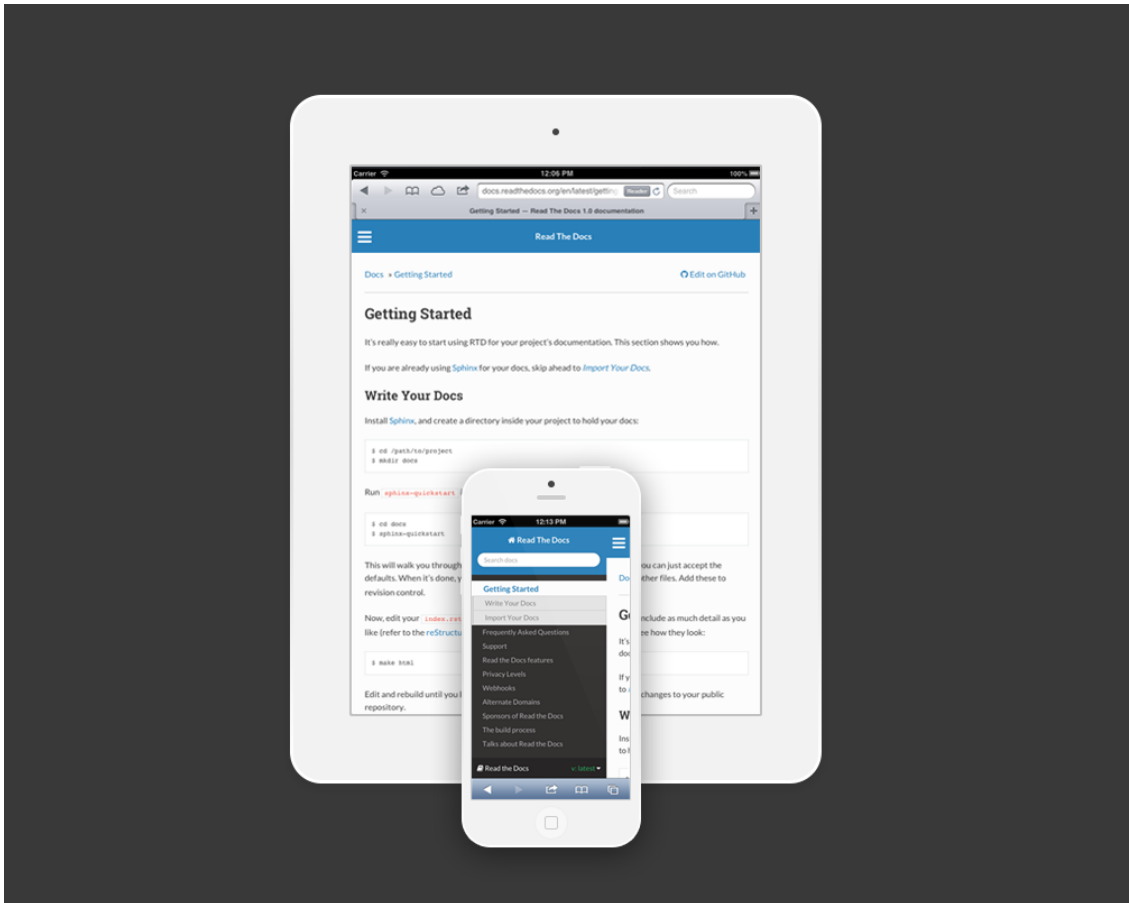
## II-B-3 - Quelques templates supplémentaires

Voici quelques exemples des templates tiers les plus populaires pour Sphinx-doc.

### Read The Doc

[https://github.com/snide/sphinx\\_rtd\\_theme](https://github.com/snide/sphinx_rtd_theme)

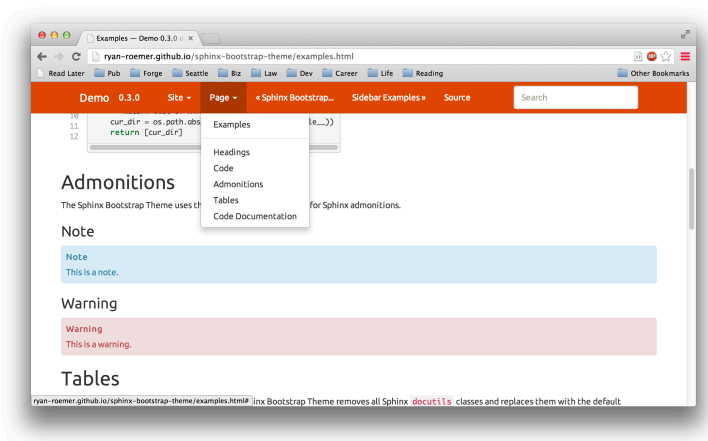
Probablement le plus utilisé, grâce entre autres au serveur Read The Doc permettant de centraliser l'ensemble de vos documentations Sphinx-doc.



## Bootstrap

<https://github.com/ryan-roemer/sphinx-bootstrap-theme>

Un autre thème assez répandu et ayant donné naissance à de nombreux dérivés.



## II-C - Format de sortie

Les formats de sorties proposés sont multiples. Voici les principaux (liste non exhaustive) :

Nom du format	Description
html	Constructeur par défaut, il permet de générer de la documentation au format HTML. On peut entre autres s'en servir pour générer un petit site web.
htmlhelp	Permet de générer un fichier d'aide CHM.
qthelp	Permet de générer un fichier d'aide QT.
devhelp	Permet de générer un fichier d'aide GNOME.
latex	Permet de générer des fichiers LaTeX, qui peuvent servir eux-mêmes à créer un pdf.
man	Permet de créer une page du manuel d'aide.
epub	Permet de générer un fichier epub.
changes	Permet de générer un fichier des changements.
pickle/json	Permet de générer un fichier Json.
xml	Permet de générer un fichier xml.

### III - Structure

À la création d'un projet, vous avez la possibilité de séparer les fichiers source des fichiers générés. C'est une option que je vous recommande d'utiliser, car elle permet de bien compartimenter les différents fichiers. C'est la structure qui sera adoptée pour la suite.

Dans le dossier source, vous trouverez alors de base quatre éléments.

#### III-A - \_static

Le dossier « \_static » est le dossier par défaut contenant les images qui seront utilisées dans la documentation.

Pour une compatibilité optimale, je vous conseille le format PNG, qui permet entre autres la transparence.

#### III-B - \_templates

Ce dossier est celui qui contiendra vos templates supplémentaires si besoin.

#### III-C - index.rst

C'est le fichier d'index, votre page de démarrage en quelque sorte. L'extension rst correspond au fichier ReST.

Nous reviendrons sur son fonctionnement, une fois les bases du langage ReST passées en revue.

#### III-D - conf.py

Le fichier conf.py est le fichier contenant toute la configuration du projet de documentation : nom du projet, version, auteur, où trouver les images, log, quelles sont les options activées...



L'ensemble des éléments pouvant être paramétrés sont listés **ICI**. Nous allons passer en revue les principaux et plus utiles.

Dans le fichier conf.py, ils sont presque tous présents, mais commentés. Pour en activer et modifier un, il vous suffit donc de le décommenter.

### III-D-1 - Langue

Plus de 30 langues sont actuellement prises en charge par Sphinx-doc. Cela ne concerne cependant que le texte généré automatiquement par l'outil. Tout texte que vous saisirez devra être dans la bonne langue.

```
1. language = 'fr'
```

### III-D-2 - Configuration HTML

Il permet d'indiquer dans quel dossier chercher un template supplémentaire. Par défaut, on utilise généralement « \_templates ».

- `templates_path = ['_templates']`
- `html_theme = '<nom du thème dans le dossier _templates>'`
- `html_theme_options = {}` # options de personnalisation de template, dictionnaire
- `html_title = « Long Title »`
- `html_short_title = « Short Title »`
- `html_logo = « _static\logo.png »`
- `html_favicon = « _static\favicon.png »`
- `html_static_path = ['_static']`

## IV - Le langage ReST

### IV-A - historique

ReST, pour Restructured Text est en réalité non pas un nouveau langage à part entière, mais une évolution de langages plus anciens.

SeText et Structured Text furent créés respectivement en 1992 et 1993. Ces langages permirent la naissance du ReST, qui vit le jour dans le but d'uniformiser la façon de coder et de corriger les défauts de ses prédécesseurs.

Son nom et surtout son acronyme visent à rappeler ses origines.

Utilisé depuis le début de XXI<sup>e</sup> siècle par la communauté Python, il est depuis 2008 partie intégrante de la communauté, notamment avec son rattachement à Sphinx-doc.

### IV-B - Principe

ReST repose sur un principe très simple : appliquer les principes de la programmation à l'écriture de documentation.

En d'autres termes, imposer un certain formalisme afin de pouvoir générer n'importe quel type de documentation.

De plus, au fil du temps, certains mots-clés, des keywords, supplémentaires sont venus s'ajouter aux possibilités offertes par ReST, lui conférant la possibilité de s'interfacer avec certains langages de programmation par exemple.

## IV-C - Fonctionnement

Quelle que soit l'action que vous désirez réaliser avec ReST, certaines règles seront toujours valables. Ce sont ces règles que nous allons voir dans cette partie.

### IV-C-1 - Les directives

Le fonctionnement des directives repose sur le même principe que les blocs de code en Python : l'indentation.

ReST fonctionne sur des lignes dont les trois premiers caractères sont significatifs. Ils lui permettent de déterminer s'il s'agit de texte simple, d'une directive, ou de texte rattaché à une directive.

Dans le premier cas, on commence à écrire dès le début de la ligne.

Dans le deuxième cas, on saisira deux points, suivis d'un espace.

Enfin, dans le troisième cas, on se décalera toujours de trois espaces. Tant que vous respecterez ces trois espaces, alors vous resterez dans le bloc.

Voici à quoi ressemble une directive :

```
1. .. <nom>:: <arguments>
2.    :<options>: <valeur>
3.    <ligne vide éventuelle, parfois obligatoire>
4.    <le texte associé>
```

Les directives et leurs options (facultatives) doivent être collées (pas de ligne vide de séparation), mais doivent être séparées par des lignes vides de toute autre directive ou texte.

Cette dernière obligation est également valide pour les titres et les listes.

Ces directives peuvent être liées aux images, à des annotations, à du code... Grâce aux deux points successifs, vous les reconnaîtrez immédiatement.



*Les directives peuvent s'imbriquer les unes dans les autres, même si cela n'est pas recommandé.*

### IV-C-2 - Le fichier index.rst

Dans le fichier index.rst, sous le TOC, après trois espaces, marquez le nom d'un fichier rst

```
1.    <nom>
```

On devra alors avoir un fichier <nom>.rst dans le dossier.

Le fichier index.rst est le sommaire, en quelque sorte, de votre documentation. Il utilise la directive « toctree » qui lui permet d'ajouter un index avec lien. Son option la plus couramment utilisée est « maxdepth », qui permet d'indiquer le niveau de titre maximal à utiliser.



*Le template utilisé peut surcharger cette option.*

L'ensemble des pages présentes dans ce sommaire sera ajouté au rendu final.

Pour ajouter une page à votre sommaire, ajoutez le nom de votre fichier (sans le « .rst »), après trois espaces. Il ne faut saisir qu'un seul nom par ligne.

Exemple :

```

1. .. toctree::
2.     :maxdepth: 2
3.
4.     Page0
5.     Page1
  
```



*La directive toctree peut être utilisée dans n'importe quelle page, et pas seulement la page index.*



*À la création du fichier index.rst, vous pouvez voir une section intitulée Indices and tables (en anglais). Cette section ne sera pas abordée dans ce tutoriel. Si vous êtes curieux, je vous invite à visiter [cette page](#).*

## IV-C-3 - Espaces et sauts de ligne

ReST gère automatiquement les espaces et les sauts de ligne. Si vous saisissez plusieurs espaces à suivre, un seul apparaîtra en sortie.

Concernant les sauts de ligne, il faut laisser une ligne vide entre chaque bloc de texte. Si ce n'est pas le cas (simple retour à la ligne), alors ReST considérera qu'il est lié au précédent et adaptera la mise en forme (mise en gras par exemple).

## IV-D - Mise en forme du texte

### IV-D-1 - Gras

Pour mettre du texte en gras, il faut utiliser les caractères « \*\* ».

Saisie ReST	Rendu final
1. <b>**Exemple de texte en gras**</b>	<b>Exemple de texte en gras</b>

### IV-D-2 - Italique

Pour mettre du texte en italique, il faut utiliser le caractère « \* ».

Saisie ReST	Rendu final
1. <b>*Exemple de texte en italique*</b>	<i>Exemple de texte en italique</i>



*Il est impossible de mixer les solutions ci-dessus. Vous ne pouvez utiliser qu'une seule option à la fois. De même, il n'est pas possible de souligner du texte, cela étant réservé aux liens.*

Pour utiliser la mise en forme de texte au sein d'un texte, vous devez le faire précéder et suivre de « \ » (backslash + espace).



Saisie ReST	Rendu final
1. Exemple de texte avec un \ <b>mot</b> \ en gras	Exemple de texte avec un <b>mot</b> en gras

## IV-E - Titre

En ReST, les titres sont soulignés d'un caractère spécial indiquant leur niveau. Un même titre doit systématiquement être souligné du même caractère. Il doit y avoir autant de caractères spéciaux que de caractères dans le titre.

Vous pouvez considérer qu'il existe cinq niveaux principaux pour les titres. Cela permet de couvrir en général l'ensemble du besoin.

Niveau	Caractère à utiliser
1	*
2	=
3	-
4	~
5	+



Il existe en réalité plus de niveaux, mais ceux présentés ici suffisent dans la majorité des cas.



Le titre de page peut être souligné ET surligné de son caractère spécial « \* ».

Saisie ReST	Rendu final
1. ***** 2. Titre de page 3. ***** 4. 5. Titre de niveau 1 6. ***** 7. 8. Titre de niveau 2 9. ===== 10. 11. Titre de niveau 3 12. ----- 13. 14. Titre de niveau 4 15. ~~~~~ 16. 17. Titre de niveau 5 18. ++++++	<div>Titre de page</div> <div>Titre de niveau 1</div> <div>Titre de niveau 2</div> <div>Titre de niveau 3</div> <div>Titre de niveau 4</div> <div>Titre de niveau 5</div>

## IV-F - Échappement

Saisie ReST	Rendu final
1. On échappe les caractères spéciaux avec <code>\\</code>	On echappe les caracteres speciaux avec <code>\</code>
1. On peut couper une ligne de source <code>\</code> 2. de cette manière, pour une meilleure lecture.	On peut couper une ligne de source de cette maniere, pour une meilleure lecture.
1. Voici comment insérer du texte sans qu'il soit interprété : les caractères spéciaux sont <code>``\n, \r, \t``</code>	Voici comment inserer du texte sans qu'il soit interpreter : les caracteres speciaux sont <code>\n, \r, \t</code>

## IV-G - Commentaires

Saisie ReST	Rendu final
1. ***** 2. Titre de page 3. ***** 4. 5. .. Ceci est un commentaire (« . » + « . » + espace)	<div>Titre de page</div>

## IV-H - Liste

### IV-H-1 - Classique

Saisie ReST	Rendu final
1. Liste classique 2. 3. * ligne 1 4. * ligne 2 5. * ligne 3	<div>Liste classique</div> <ul style="list-style-type: none"> <li>• ligne 1</li> <li>• ligne 2</li> <li>• ligne 3</li> </ul>

### IV-H-2 - Numérotée

Saisie ReST	Rendu final
1. Liste numérotée manuelle 2. 1. ligne 1 3. 2. ligne 2 4. 3. ligne 3	<div>Liste numerotee manuelle</div> <ol style="list-style-type: none"> <li>1. ligne 1</li> <li>2. ligne 2</li> <li>3. ligne 3</li> </ol>
1. Liste numérotée automatique 2. #. ligne 1 3. #. ligne 2 4. #. ligne 3	<div>Liste numerotee automatique</div> <ol style="list-style-type: none"> <li>1. ligne 1</li> <li>2. ligne 2</li> <li>3. ligne 3</li> </ol>

## IV-I - Tableaux

Il existe trois principales méthodes pour faire des tableaux en ReST.

### IV-I-1 - Méthode 1

Saisie ReST	Rendu final
<pre> 1. +-----+-----+ 2.   T0   T1   3. +=====+=====+ 4.   x0   x1   x2   5. +-----+-----+ </pre>	
<pre> 1. ===== 2. T0 T1 3. ----- 4. T01 T02 T11 5. ===== 6. S1 S2 S3 7. S4 S5 S6 8. ===== </pre>	

### IV-I-2 - Méthode 2

Saisie ReST	Rendu final
<pre> 1. .. csv-table:: Tableau de    demo 2.   :header: « Nom »,    « mail » 3.   :widths: 40, 40 4. 5.   « Dupond », « Martin » 6.   « Durand », « Eric » </pre>	

La méthode 2 présente l'avantage de permettre de fixer la largeur des colonnes. En contrepartie, vous ne pourrez fusionner des cellules.

### IV-I-3 - Méthode 3

Saisie ReST	Rendu final
<pre> 1. .. list-table:: Exemple de    tableau 2.   :widths: 10 10 20 3.   :header-rows: 1 4.   :stub-columns: 1 5. 6.   * - Titre 1 7.     - Titre 2 8.     - Titre 3 9.   * - Contenu 1 10.    - Contenu 2 11.    - Contenu 3 </pre>	

Cette dernière méthode est certainement la plus pratique et la plus souple d'usage. Je la recommande donc.

L'option «:header-rows : » permet de fixer la barre de titre horizontale.

L'option «:stub-columns : » permet de fixer la barre de titre verticale.

## IV-J - Insertion de bloc

### IV-J-1 - Texte/code depuis un fichier

Saisie ReST	Rendu final
<pre> 1. ***** 2. Titre de page 3. ***** 4. 5. .. literalinclude:: _static/    exemple.py </pre>	<div>Titre de page</div> <pre> """ ceci est le contenu de mon fichier python. Sphinx detecte l'extension et s'adapte au langage si ce n'est pas un simple fichier texte""" </pre>
<pre> 1. .. literalinclude:: _static/    exemple.txt 2.     :encoding: latin-1 3.     :start-after: debut 4.     :end-before: fin </pre>	<div>Titre de page</div> <pre> """ ceci est le contenu de mon fichier python. Sphinx detecte l'extension et s'adapte au langage si ce n'est pas un simple fichier texte""" </pre>
<pre> 1. .. literalinclude:: _static/    exemple.txt 2.     :encoding: latin-1 3.     :lines: 1,3-4 </pre>	<div>Titre de page</div> <pre> debut Sphinx detecte l'extension et s'adapte au langage si ce n'est pas un simple fichier </pre>

### IV-J-2 - Code directement saisi

Saisie ReST	Rendu final
<pre> 1. ***** 2. Titre de page 3. ***** 4. 5. .. code-block:: python 6. 7.     print(« hello world ») </pre>	<div>Titre de page</div> <pre> print(" hello world ") </pre>

Pour la coloration syntaxique, Sphinx-doc utilise « pygments ». Grâce à cet outil, voici les principaux langages (liste non exhaustive) que vous pourrez utiliser dans Sphinx-doc.

ActionScript	Gettext catalogs	PostScript
Ada	Gherkin	POV-Ray scenes
ANTLR	GL shaders	PovRay
Apache config files	Gnuplot script	PowerShell
AppleScript	Groff markup	Prolog
Assembly	Groovy	Python 2.x and 3.x
Asymptote	Haskell	Ragel
Awk	HTML	Rebol
Bash shell scripts	HTTP sessions	Redcode
BBCode	IDL	Redcode
Befunge	INI-style config files	ReST
Boo	Io	Robot Framework
BrainFuck	IRC logs	RPM spec files
C, C++	Java	Ruby
C#	JavaScript	Rust
Cheetah templates	JSP	S, S-Plus, R
Clojure	Lighttpd config files	Scala
CMake	LLVM	Scheme
CoffeeScript	Logtalk	Scilab
ColdFusion	Lua	Smalltalk
Common Lisp	Makefiles	Smarty templates
Coq	Mako	SNOBOL
CSS	Matlab	SQL, also MySQL, SQLite
Cython	MiniD	Squid configuration
D	Modelica	Tcl
Dart	Modula-2	tcsh
Debian control files	MoinMoin/Trac markup	Tea
Delphi	MuPad	TeX
Diff files	Myghty	Vala
Django / Jinja templates	MySQL	Verilog
DTD	Nemerle	VHDL
Dylan	Nginx config files	Vim Script
ERB	Nimrod	Visual Basic.NET
Erlang	Objective-C	Visual FoxPro
F#	Objective-J	Windows batch files
Factor	OCaml	XML
Fancy	Octave	XQuery
Fortran	Perl	XSLT
Genshi	PHP	YAML

Pour trouver le mot-clé à utiliser, il vous faudra consulter [la page des « lexer »](#). Un Lexer est simplement une sorte de traducteur permettant la coloration syntaxique.

Par exemple, pour Python, cette page vous indique ce qui suit :

```
class pygments.lexers.agile.PythonLexer
```

Short names:python, py, sage

Filenames:\*.py, \*.pyw, \*.sc, SConstruct, SConscript, \*.tac, \*.sage

MIME types:text/x-python, application/x-python



For Python source code.

Ce qui est déclaré en tant que « shortname » est le mot-clé à utiliser dans Sphinx-doc, dans notre cas : « python ».


## IV-K - Lien Hypertexte et de téléchargement

Saisie ReST	Rendu final
1. <code>`Python &lt;http://www.python.org/&gt;`_</code>	<a href="http://www.python.org/">Python</a>
1. <code>:download: `Mon logiciel &lt;LINK&gt;`_</code>	<b>download:</b> <a href="#">Mon logiciel</a>



Remplacez *LINK* par le lien web de votre choix. De plus, il faut remplacer les espaces par %20 et les & par &amp;.

## IV-L - Image

Saisie ReST	Rendu final
1. <code>*****</code> 2. <code>Titre de page</code> 3. <code>*****</code> 4. <code></code> 5. <code>.. image:: _static/tux.png</code> 6. <code>:align: center</code>	Titre de page 



Vous pouvez remplacer *center* par *left* (par défaut) ou encore *right*. De même, il existe l'option « `:scale : 50%` » qui vous permet de gérer la taille de l'image.

## IV-M - Boîte

Nous allons voir ici comment insérer ce qu'on appelle des boîtes. Il s'agit en réalité de Postit, en quelque sorte, que l'on peut insérer où on le désire dans la documentation.

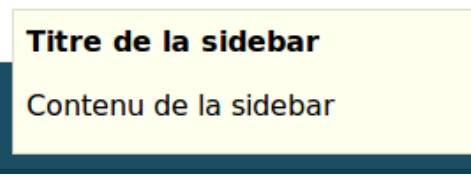
### IV-M-1 - Voir aussi

Saisie ReST	Rendu final
1. <code>.. seealso:: Ceci est une boîte « voir aussi »</code>	<b>See also:</b> Ceci est une boîte "voir aussi"

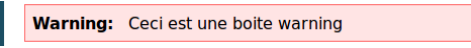
### IV-M-2 - Note

Saisie ReST	Rendu final
1. <code>.. note:: Ceci est une boîte « Note »</code>	<b>Note:</b> Ceci est une boîte "Note"


## IV-M-3 - Note de côté

Saisie ReST	Rendu final
<pre>1. .. sidebar:: Titre de la sidebar 2. 3.     Contenu de la sidebar</pre>	

## IV-M-4 - Attention

Saisie ReST	Rendu final
<pre>1. .. warning:: Ceci est une    boite warning</pre>	

## IV-M-5 - A faire

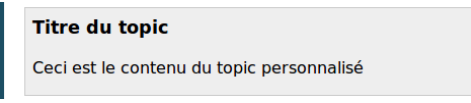
Saisie ReST	Rendu final
<pre>1. .. todo:: Liste de choses à    faire</pre>	

Pour afficher une liste à faire, il faut modifier le paramétrage dans `conf.py`. Ajouter « `'sphinx.ext.todo'` » à la liste « `extension` », puis les lignes suivantes :




```
1. [extensions]
2. todo_include_todos=True
```

## IV-M-6 - Personnaliser

Saisie ReST	Rendu final
<pre>1. .. topic:: Titre du topic 2. 3.     Ceci est le contenu du    topic personnalisé</pre>	

## IV-N - Substitution

Saisie ReST	Rendu final
<pre>1. ***** 2. Titre de page 3. ***** 4. 5. ..  alias0  replace::    Batman 6. 7. Dans cette histoire,      alias0  est un personnage    connu.  alias0  est également    un héros.</pre>	



Faites bien attention à ce que vos « `|alias|` » soient bien entourés d'espaces. Sinon, ils ne seront pas interprétés comme il le faut.

## IV-O - Note de bas de page

Saisie ReST	Rendu final
<pre> 1. ***** 2. Titre de page 3. ***** 4. 5. Introduction aux notes de    bas de page 6. 7. NB0 [#f1]_ NB2 [#f2]_ 8. 9. Et une petite dernière 10. 11. NB3 [#f3]_ 12. 13. 14. .. rubric:: Footnotes 15. 16. .. [#f1] Nota Bene 1 17. .. [#f2] Nota Bene 2 18. .. [#f3] Nota Bene 3 </pre>	<p>Titre de page</p> <p>Introduction aux notes de bas de page</p> <p>NB0 [1] NB2 [2]</p> <p>Et une petite dernière</p> <p>NB3 [3]</p> <p><b>Footnotes</b></p> <p>[1] Nota Bene 1</p> <p>[2] Nota Bene 2</p> <p>[3] Nota Bene 3</p>

## IV-P - Citation

Saisie ReST	Rendu final
<pre> 1. ***** 2. Titre de page 3. ***** 4. 5. « to be or not to be » [1]_    est une citation connue. 6. 7. 8. .. [1] du célèbre    Shakespeare </pre>	<p>Titre de page</p> <p>“to be or not to be” [1] est une citation connue.</p> <p>[1] du celebre Shakespeare</p>



La différence entre citation et Note de bas de page tient principalement au fait que les notes de bas de page sont à numérotation automatique.

## IV-Q - Expression mathématique avec pngmath

Le module pngmath vous permet de saisir des formules mathématiques, au format LaTeX, et de les afficher sous la forme d'images. La directive est « `math` ».

Nous allons ici voir les principes de base. Pour des informations plus complètes, je vous renvoie vers ces liens : [LIEN1](#), [LIEN2](#), [LIEN3](#).

Il faut séparer les différentes lignes de formules par une ligne vide. Toutes lignes écrites les unes à la suite des autres (sans ligne vide) seront considérées comme complémentaires. Par conséquent un alignement automatique sera effectué.

Les groupes sont contenus dans des accolades. Par groupe, il faut comprendre des numérateurs par exemple.

Tout caractère de délimitation (parenthèse, crochet, accolade) doit être échappé avec le caractère « \ ».

Désignation mathématique	Écriture LaTeX
Puissance	$a^2$
Indice	$a_2$
Mix	$a^2_{\{3\}}$
Fraction	$\frac{\text{numérateur}}{\text{denominateur}}$
Racine carrée	$\sqrt[\text{puissance}]{\text{valeur}}$
Sinus, cosinus, tangente	$\sin, \cos, \tan, \arcsin, \arccos, \arctan$
Somme	$\sum_{\text{valeur inférieure}}^{\text{valeur supérieure}}$
Intégrale	$\int_{\text{valeur inférieure}}^{\text{valeur supérieure}}$
Infini	$\infty$
Pi	$\pi$

Saisie ReST	Rendu final
<pre>1. ***** 2. Titre de page 3. ***** 4. 5. .. math:: 6.     \sum_{a}^b x_n = c_1 +    c_2^3 + \cos(\pi)</pre>	<div>Titre de page</div> $\sum_{n=1}^N x_n = c_1 + c_2^3 + \cos(\pi)$



Pour utiliser cette fonctionnalité, il faut l'avoir activée à la création du projet de documentation. Pour l'activer après la création, je vous renvoie vers la documentation officielle.



Il ne faut pas oublier d'installer LaTeX & dvipng, sans quoi, le rendu en image ne sera pas effectué. Sous Linux, ajouter texlive-latex-extra (et ses dépendances) peut corriger des erreurs complémentaires.

## IV-R - Documentation Python manuelle

Si vous le désirez, vous avez la possibilité de créer la documentation de votre code à la main. Pour cela, de simples directives existent. Nous allons voir ici les principales.

Directive	Description
<pre>1. .. py:module:: name</pre>	Permet de définir un module ou un package.
<pre>1. .. py:class::    name(parameters)</pre>	Permet de définir une classe.
<pre>1. .. py:function::    name(parameters)</pre>	Permet de définir une fonction.
<pre>1. .. py:method::    name(parameters)</pre>	Permet de définir une méthode.
<pre>1. .. py:attribute:: name</pre>	Permet de définir un attribut.
<pre>1. .. py:decorator::    name(parameters)</pre>	Permet de définir un décorateur.
<pre>1. .. py:exception:: name</pre>	Permet de définir une exception.

Pour les paramètres complémentaires, ils sont communs avec la section [IV.S.3](#). L'exemple en [VI](#) vous montrera comment mettre en application pratique les directives vues ci-dessus.

## IV-S - Documentation Python automatique

Pour pouvoir autodocumenter son code, il faut agir à trois niveaux :

- `conf.py` ;
- les fichiers `.rst` ;
- les docstrings.

De plus, il faut utiliser quelques commandes ReST spécifiques pour l'autodocumentation Python . Nous allons passer en revue les principales d'entre elles.

### IV-S-1 - Indiquer à Sphinx-doc où trouver le code Python

Cela se passe dans le fichier « `conf.py` ».

Par défaut, il y a une ligne commentée : « `sys.path.insert(0, os.path.abspath('.'))` »

Il faut décommenter cette ligne, puis remplacer le chemin actuel « `.` » par le chemin où se trouve votre code.

Bien entendu, si la documentation est dans le dossier de votre code, vous pouvez effectuer un « `../..` ».

De même, vous pouvez définir plusieurs chemins, mais toujours avec `os.path.abspath()`.

### IV-S-2 - Côté fichiers `.rst`

Dans un fichier `.rst`, vous avez la possibilité d'écrire comme dans tout autre fichier `.rst`, en plus d'y ajouter la documentation automatique de votre code.

Ainsi, quand vous désirez ajouter une documentation précise (module, classe...) vous avez juste à saisir la directive adéquate, et préciser ses éventuelles options.

Chaque élément (module, fonctions...) est appelé « membre ».

Saisie ReST	signification
<code>1. .. automodule::     nom_du_module</code>	Indique quel module documenter. Existent aussi en autoclass, autoexception.
<code>1.     :members:</code>	Option : fait apparaître les éléments publics (qui ne commencent pas par « <code>_</code> »). Si on ajoute des noms, ne prend que les membres nommés, sinon prend tout.
<code>1.     :special-members:</code>	Option : permet d'indiquer de stipuler les constructeurs dans le cas des éléments publics seulement.
<code>1.     :undoc-members:</code>	Option : permet d'afficher également dans la documentation les membres sans docstring.
<code>1.     :private-members:</code>	Option : montre les éléments privés en plus.
<code>1.     :show-inheritance:</code>	Option : montre les classes mères dont héritent les classes documentées.

Saisie ReST	Rendu final
<code>1. .. autofunction:: nom</code>	Documente juste une fonction. Ne possède pas d'option.

## IV-S-3 - Côté docstrings

### Dans les docstrings des modules/packages

Ce qui suit ne concerne que la docstring pour les modules et les packages.

Saisie ReST	Rendu final
<code>1. .. module:: nom</code>	Indique le début de la description d'un module, package ou sous module.
<code>1. :platform: Unix, Windows, Mac</code>	Indique la plate-forme cible. Seules trois valeurs sont admises.
<code>1. :synopsis: resume</code>	Permet d'ajouter un résumé rapide.
<code>1. ..moduleauthor:: premier &lt;premier@o.fr&gt;</code>	Permet d'indiquer le nom de l'auteur et son mail. Il est possible de définir plusieurs auteurs. Il faut alors un « moduleauthor » par auteur.

### Dans les autres docstrings

En plus de texte explicatif de la docstring, on peut utiliser les tags du tableau ci-dessous. Les paramètres, arguments... sont appelés des « éléments ».

Saisie ReST	Rendu final
<code>1. :param nom_param: description</code>	Permet de décrire un paramètre. Une description par paramètre.
<code>1. :arg nom_arg: description</code>	Permet de décrire un argument. Correspond au args de « *args et *kwargs ».
<code>1. :key nom_key: description</code>	Correspond au kwargs de « *args et *kwargs ».
<code>1. :var nom_variable: description</code>	Permet de décrire une variable.
<code>1. :type element: description</code>	Permet de décrire le type de l'élément. Bien qu'on puisse inscrire ce que l'on veut, il faut utiliser autant que possible les valeurs suivantes :Callable, int, float, long, str, tuple, list, dict, None, True, False, boolean.
<code>1. :returns: description</code>	Permet de décrire ce qui est retourné. Va de pair avec:rtype:
<code>1. :rtype: int</code>	Permet de définir le type de la variable retournée.
<code>1. :raises nom_exception: description</code>	Permet de préciser une exception levée. Pour en déclarer plusieurs, il faut alors une ligne par exception.



À l'intérieur des docstrings, les sauts de ligne apparaîtront dans l'autodocumentation.



*Vous pouvez utiliser plusieurs fois à suivre le couple « :return:/:rtype: » si vous retournez plusieurs éléments.*

## V - Pratique : créer une simple documentation

Je vous recommande de compartimenter au maximum votre documentation, pour des raisons de maintenabilité et d'aisance de lecture. Pour cela, externalisez au maximum le code et toute référence textuelle spécifique (1 fichier/cas).

Pour mieux structurer mes documentations, je crée un dossier « `_code` » pour mes fichiers source et un dossier « `_refs` » pour mes références diverses (texte principalement, dans des fichiers txt).

### V-A - Installation de Sphinx-doc

Toute la procédure d'installation sous les différents OS est détaillée sur la [page dédiée officielle](#) de Sphinx-doc.

Chaque cas étant différent, je vous invite à consulter cette page afin de trouver la méthode adéquate à votre cas.

### V-B - Création d'un nouveau projet

Pour créer un nouveau projet, ouvrez un terminal, ou une commande DOS, placez-vous dans le dossier où vous désirez créer votre documentation (cas échéant, créez un dossier dédié avant de continuer) et lancez la commande « sphinx-quickstart ».

Commence alors l'interrogatoire afin de créer la structure de votre documentation. Chaque question indiquera sa valeur par défaut (en cas d'un appui direct sur entrée), entre crochets.



*Sphinx-Quickstart est exclusivement en anglais. De plus, il ne fait pas de distinction Linux, Windows, Mac. Ne soyez donc pas étonné de voir des questions relatives à un autre OS.*

#### >Root path for the documentation

Indiquez ou créez la documentation. Par défaut, l'emplacement où vous vous trouvez.

#### >Separate source and build directories (y/N)

Répondez toujours oui (y) à cette question, toujours dans le but de bien compartimenter votre code. Cela créera deux dossiers : un pour les sources, et un pour les builds.

#### >Name prefix for templates and static dir

Permet de stipuler le préfixe pour vos dossiers et fichiers rattachés à Sphinx-doc. Je vous conseille fortement de laisser ce paramètre à sa valeur par défaut : « `_` ».

#### >Project name

Saisissez ici le nom de votre projet, en respectant la casse.

#### >Author name(s)

Saisissez ici les noms des auteurs.

#### **>Project version**

Indiquez ici la version de votre projet.

#### **>Project release**

Indiquez ici la release de votre version. Par défaut à la même valeur que la version.

#### **>Source file suffix**

Sert à préciser le suffixe des fichiers sphinx-doc. Je vous conseille de laisser ce paramètre à sa valeur par défaut : « .rst ».

#### **>Name of your master document (without suffix)**

Permet de préciser comment s'appellera la première page de votre documentation. Je vous conseille, là aussi de laisser le paramètre à sa valeur par défaut : « index ».

#### **>Do you want to use the epub builder (y/N)**

Si vous désirez pouvoir générer des epub, placez ce paramètre à oui.

#### **>autodoc: automatically insert docstrings from modules (y/N)**

Ce paramètre doit être placé à oui, afin de pouvoir documenter du code Python.

#### **>doctest: automatically test code snippets in doctest blocks (y/N)**

Je n'ai trouvé aucune info sur ce que fait vraiment ce paramètre.

#### **>intersphinx: link between Sphinx documentation of different projects (y/N)**

Ce paramètre vous permet de lier plusieurs documentations ensemble.

#### **>todo: write « todo » entries that can be shown or hidden on build (y/N)**

Si vous désirez faire apparaître les boîtes « todo » dans le rendu, alors renseignez ce paramètre à oui.

#### **>coverage: checks for documentation coverage (y/N)**

Ce paramètre permet de générer des statistiques sur la couverture de code, d'un point de vue documentation.

#### **>pngmath: include math, rendered as PNG images (y/N)**

Pour pouvoir insérer des formules mathématiques, telles que vues précédemment, placez ce paramètre à oui. Par défaut à « non ».

#### **>mathjax: include math, rendered in the browser by MathJax (y/N)**

Un substitut à pngmath, non explicité dans ce tutoriel. Par défaut à « non ».



**>ifconfig: conditional inclusion of content based on config values (y/N)**

Vous permet d'ajouter quelques contrôles de flux dans votre documentation. Au cours de mes utilisations et recherches, je n'ai jamais trouvé quelqu'un qui s'en servait. Par défaut à « non ».

**>viewcode: include links to the source code of documented Python objects (y/N)**

Ce paramètre vous permet de rajouter des liens vers votre code Python au sein de votre documentation. Par défaut à « non ».

**>Create Makefile? (Y/n)**

Génère un script gérant le rendu de la documentation dans le format désiré. Par défaut à « oui ».

**>Create Windows command file? (Y/n)**

Permet de faire l'interface entre Windows et le script Makefile. Par défaut à « oui ».

Voilà, la structure de base de votre documentation est désormais prête.

Vous disposez alors d'un sous-dossier source, contenant un dossier « \_static », et les fichiers index.rst et conf.py. Le sous-dossier build, lui, sera créé lors de votre premier rendu.

Maintenant que nous avons vu comment créer une structure de base, passons aux choses sérieuses.

## V-B-1 - Notre projet de démo

Pour cette petite démo, nous aurons, outre la page d'index, deux pages, l'une contenant un petit speech et un lien, et l'autre une formule, une image, un texte et quelques boîtes. Un tout petit projet donc, totalement bateau, mais néanmoins suffisant pour prendre l'outil en main.

Commencez donc par vous créer un nouveau projet (vous êtes libre du choix de vos divers paramètres). Une fois votre projet créé, allez dans le dossier source, et créez deux fichiers texte supplémentaires que vous nommerez « Page0.rst » et « Page1.rst » (attention à l'extension).

Ensuite, ouvrez index, Page0 et Page 1, et modifiez-les de la façon suivante, avant de les enregistrer :

**Index.rst (complétez après l'entête de votre projet).**

```
1. Bienvenue sur cette documentation
2. =====
3.
4. Contents:
5.
6. .. toctree::
7.    :maxdepth: 2
8.
9.    Page0
10.   Page1
```

**Page0.rst**

```
1. *****
2. Premiere page
3. *****
4.
5. Introduction
```

```
6. *****
7.
8. Ceci est le texte de la première page de notre demo. Il vise \
9. à introduire le fonctionnement de Sphinx-doc pour créer de la documentation.
10.
11. Lien
12. ====
13.
14. Pour plus d'informations sur Sphinx-doc, je vous invite à consulter le `site officiel
    <http://sphinx-doc.org/>`_.
```

## Page1.rst

```
1. *****
2. Seconde page
3. *****
4.
5. Formule
6. *****
7.
8. .. math::
9.     \sum_{a}^{b} x_n = c_1 + c_2^3 + \cos(\pi)
10.
11. Image
12. *****
13.
14. .. image:: _static/tux.png
15.     :align: center
16.
17. Texte
18. *****
19.
20. Voici un second texte pour notre Démo.
21.
22. .. note::
23.     J'espère que cela vous parle un peu plus que la présentation présente.
```

## V-C - Génération de notre documentation au format HTML

Maintenant que nous avons organisé notre documentation et complété les différents fichiers, il ne nous reste plus qu'à générer la documentation au format HTML.

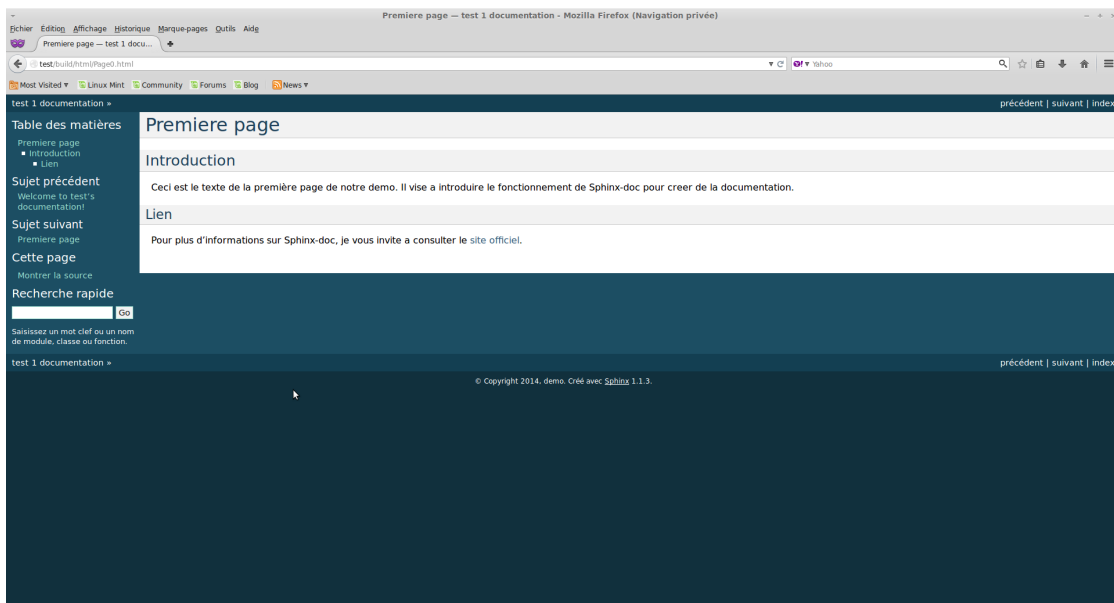
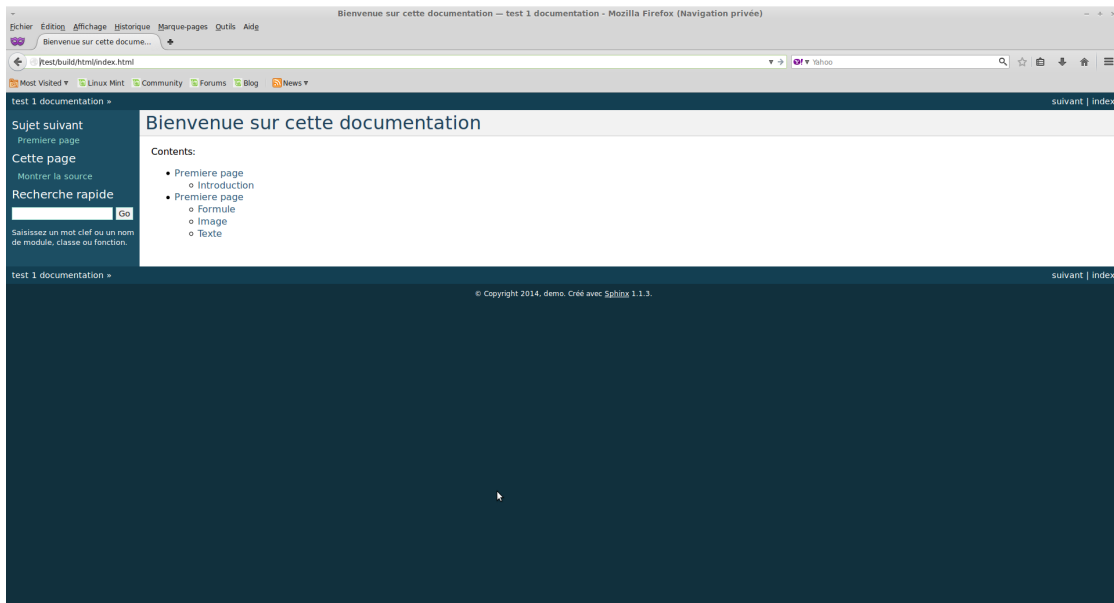
Pour cela, depuis un terminal, placez-vous à la racine de votre dossier de documentation, là où se trouve le Makefile, puis lancez la commande « make html ».

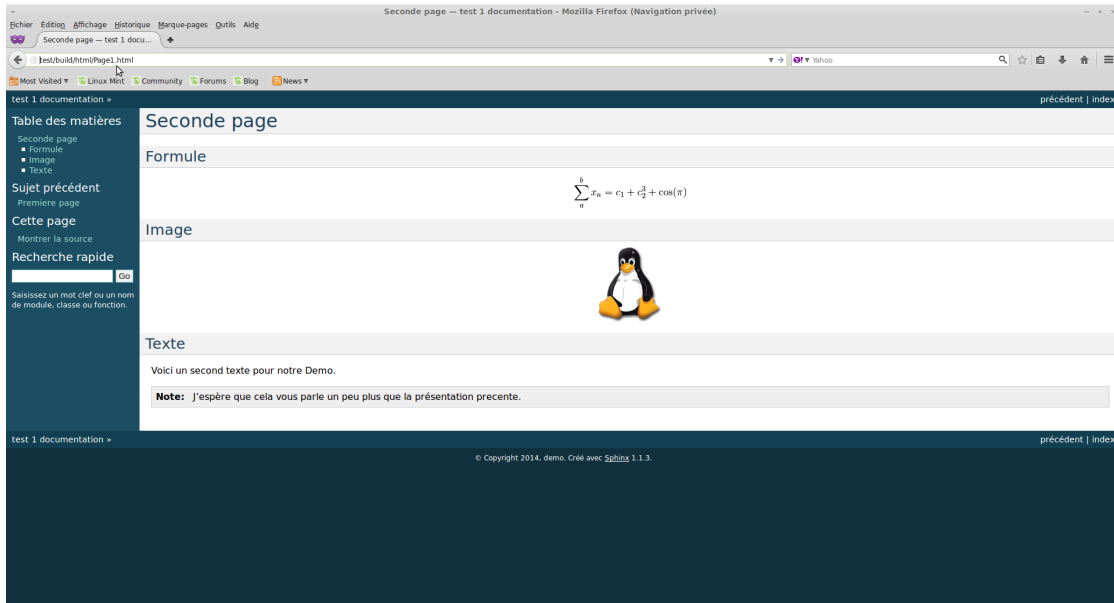
Sauf erreur, laquelle vous serait explicitement signalée, la documentation a été générée. Vous disposez alors d'un nouveau sous-dossier nommé « build », qui contient lui-même un dossier « html ». À l'intérieur de ce dossier, vous trouverez l'intégralité de votre documentation au format HTML. Lancez alors la page d'accueil (index.html par défaut).



*Si vous ajoutez/supprimez une nouvelle page, il vous faudra totalement supprimer le contenu du dossier HTML, puis régénérer toute la documentation. En effet, sans cela, l'ensemble des pages ne seront pas mises à jour et vous aurez des liens absents sur les pages déjà créées.*

## V-D - Rendu final





## VI - Pratique : autodocumenter son code Python

Maintenant que nous avons vu comment créer une documentation simple, nous allons monter en complexité, en nous intégrant avec du code source Python, et en documentant automatiquement ce code.

Je vous laisse donc créer un nouveau projet. Par contre cette fois, n'oubliez pas d'activer le paramètre « autodoc ».

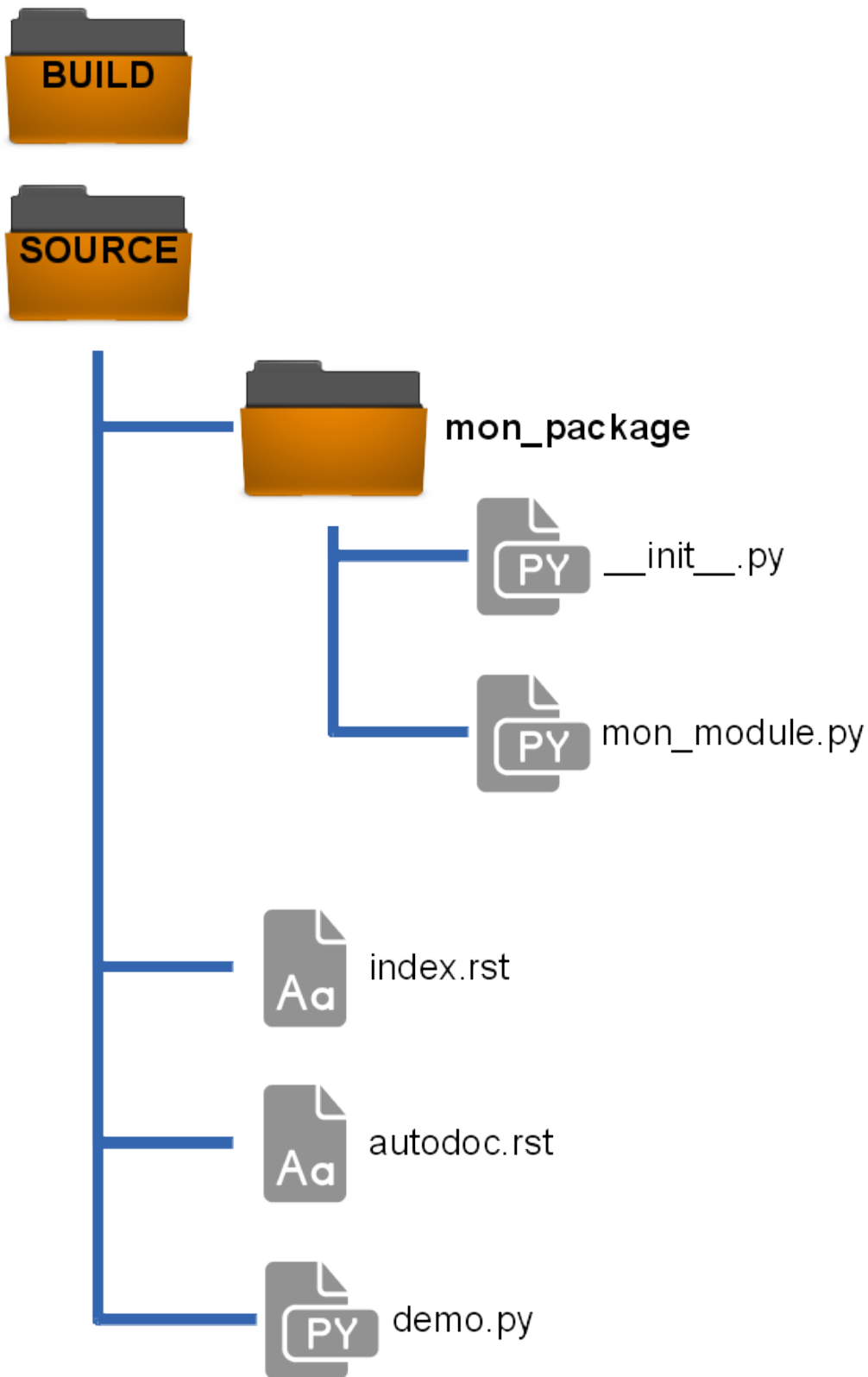
Une fois dans le dossier source, créez un fichier « autodoc.rst », puis un fichier « demo.py ». Ensuite, créez un dossier « mon\_package », et placez-vous dedans.

Maintenant, créez un fichier « \_\_init\_\_.py » et « mon\_module.py ».

Voici une représentation visuelle. « Conf.py » n'a volontairement pas été représenté. Cette vue vise à représenter uniquement les fichiers qui vont interagir dans notre cas.



*N'oubliez pas de décommenter le chemin d'accès aux sources dans le fichier conf.py.*



Éditez l'ensemble des fichiers, et renseignez-les de la façon suivante :

**Index.rst (complétez après l'entête de votre projet).**

1. Bienvenue sur cette documentation

```
2. =====
3.
4. Contents:
5.
6. .. toctree::
7.     :maxdepth: 2
8.
9.     autodoc
10.
11. Complement
12. =====
13. Comme promis voici un petit exemple de documentation manuelle
14.
15. .. py:function:: ma_fonction(arg00, arg01, arg03='4')
16.
17.     :param arg00: premier argument
18.     :type arg00: int
19.     :param arg01: second argument
20.     :type arg01: dict
21.     :param arg02: troisieme argument
22.     :type arg02: str
23.     :return: l'ensemble des elements demandes
24.     :rtype: list
```

## autodoc.rst

```
1. *****
2. Page d'autodocumentation
3. *****
4.
5. demo.py
6. =====
7.
8. Voici l'autodocumentation d'une simple fonction de module
9.
10. .. autofunction:: demo.ma_fonction_simple
11.
12. mon_package
13. =====
14.
15. Voici l'autodocumentation d'un package/module
16.
17. .. automodule:: mon_package
```

## demo.py

```
1. # -*- coding: utf-8 -*-
2.
3. def ma_fonction_simple(numerateur, denominateur):
4.     « " »
5.     Effectue une division
6.
7.     :param numerateur: le numerateur de la division
8.     :type numerateur: int
9.     :param denominateur: le denominateur de la division
10.    :type denominateur: int
11.    :return: la valeur entiere de la division
12.    :rtype: int
13.    « " »
14.
15.    if denominateur == 0:
16.        print « denominateur interdit »
17.    else:
18.        result = int(numerateur)/int(denominateur)
19.        return result
20.
21. if __name__ == « __main__ »:
22.     print ma_fonction_simple(4,2)
```

## \_\_init\_\_.py

```
1. « " »
2. .. automodule:: mon_package.mon_module
3.     :members:
4. « " »
```

Veuillez noter l'action effectuée ici. Pour que « automodule » fonctionne sur toute la chaîne du package, il faut créer des relais. C'est ce qui est effectué ici, dans le « \_\_init\_\_.py ».

De même, remarquez l'utilisation du « members ». Il permet de préciser que l'on désire une récursivité dans la documentation. Sans cela, l'autodocumentation se limiterait au seul module. Essayez de le supprimer pour voir.

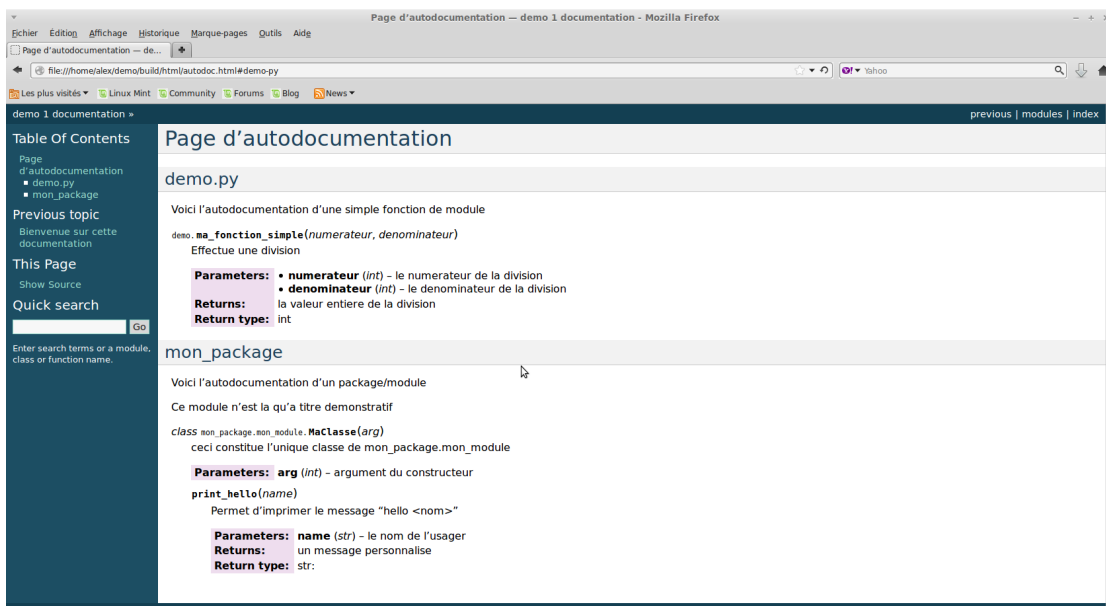
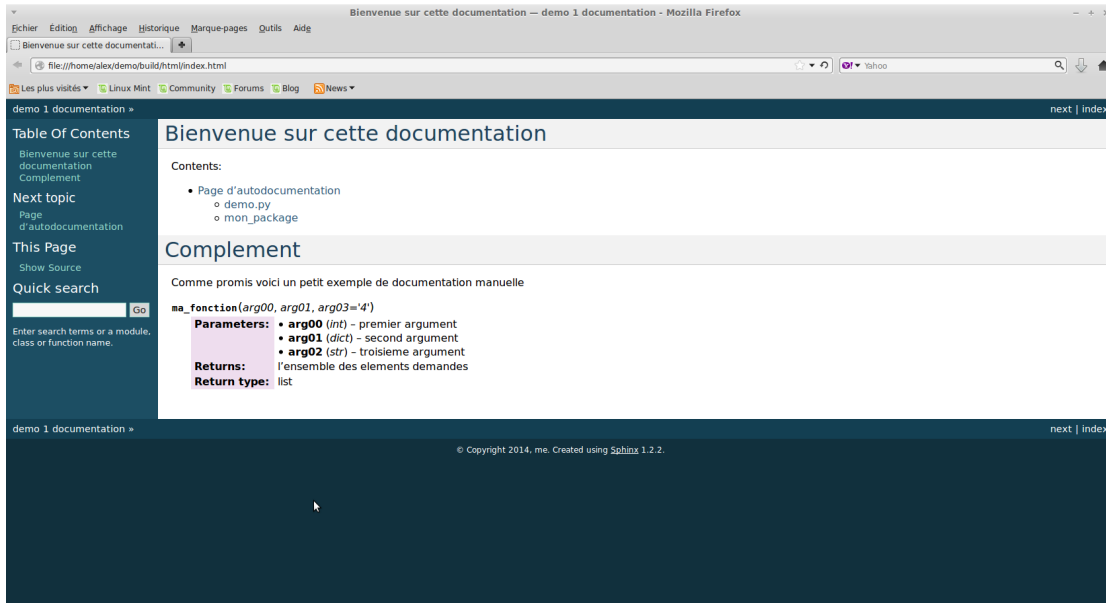
## mon\_module.py


```
1. # -*- coding: utf-8 -*-
2.
3. « " »
4. Ce module n'est la qu'a titre demonstratif
5. « " »
6.
7. class MaClasse():
8.     « " »
9.     ceci constitue l'unique classe de mon_package.mon_module
10.
11.     :param arg: argument du constructeur
12.     :type arg: int
13.     « " »
14.
15.     def __init__(self, arg):
16.         « " »
17.         Ceci est le constructeur de MaClasse
18.
19.         :param param: p1
20.         :type param: int
21.         « " »
22.         None
23.
24.     def print_hello(self, name):
25.         « " »
26.         Permet d'imprimer le message « hello <nom> »
27.
28.         :param name: le nom de l'usager
29.         :type name: str
30.         :return: un message personnalise
31.         :rtype: str:
32.         « " »
33.         print « hello » + name
34.         return « Bonjour souhaite a » + name
35.
36. if __name__ == « __main__ »:
37.     mon_objet = MaClasse()
38.     print mon_objet.print_hello(« ami lecteur »)
```

## VI-A - Génération de notre documentation au format HTML

Ici rien de sorcier, la procédure est la même que tout à l'heure.

## VI-B - Rendu final




 Veuillez remarquer que le constructeur n'est pas documenté. De fait, si vous désirez documenter le constructeur d'une classe, les directives « `:param:` », « `:type:` »... doivent alors être placées dans la docstring de la classe, et non dans le constructeur.

## VII - Conclusion

Comme nous venons de le voir, Sphinx-doc est un outil extrêmement puissant permettant d'uniformiser les outils de documentation.

Largement reconnu par la communauté Python et professionnelle, de nombreux forums, sites spécialisés... existent sur le sujet et sont sources d'assistance et d'astuces afin d'aller toujours plus loin.



De même, la documentation en ligne de Sphinx-doc est très poussée, et pour peu que vous vous donniez la peine de chercher, vous trouverez ce que vous intéresse.

Pour conclure, ce logiciel fait ainsi partie de cette catégorie d'indispensables permettant de se simplifier la vie tout en optimisant la qualité du travail effectué.

## VIII - Liens

- [https://pythonhosted.org/an\\_example\\_pypi\\_project/sphinx.html#full-code-example](https://pythonhosted.org/an_example_pypi_project/sphinx.html#full-code-example)
- <http://codeandchaos.wordpress.com/2012/07/30/sphinx-autodoc-tutorial-for-dummies/>
- <http://sphinx-doc.org/ext/autodoc.html?highlight=docstring>
- <http://sphinx-doc.org/domains.html#signatures>
- <http://danteslab-eng.blogspot.fr/2013/12/documenting-code-with-sphinx.html>
- <https://docs.python.org/3.1/documenting/markup.html>

## IX - Remerciements

Merci aux personnes suivantes pour leur aide dans la rédaction de cet article :

- **Plxpy**
- **ClaudeLELOUP**