Une introduction aux outils de documentation.

Présentation et démonstrations des outils courants de documentation et d'analyse de code.

Lilian Besson (nounou responsable : Vincent Le Gallic)

Cr@ns

ENS de Cachan

19/02/2013

Ces slides et les ressources utilisées pour les exemples sont disponibles en lignes [1] http://perso.crans.org/besson/publis/seminaire_crans/, et sur la page du Wiki Cr@ns.
N'hésitez pas à me contacter : mailto:lilian.besson@crans.org.

1. Sous licence GPLv3.

Plan

Surtout, n'hésitez pas à m'interrompre !

À propos (1)

De quoi va-t-on parler ?

Différents outils pour **documenter** du code :

1. `OCaml` ;
2. `Python`.

Et aussi . . .

Outils **complémentaires** d'*analyse statique* et d'aide au développement.

À propos (1)

De quoi va-t-on parler ?

Différents outils pour **documenter** du code :

1. `OCaml` ;
2. `Python`.

Et aussi . . .

Outils **complémentaires** d'*analyse statique* et d'aide au développement.

À propos (1)

De quoi va-t-on parler ?

Différents outils pour **documenter** du code :

1. `OCaml` ;
2. `Python`.

Et aussi . . .

Outils **complémentaires** d'*analyse statique* et d'aide au développement.

À propos (2)

Une simple définition

Ajouter des informations supplémentaires :

1. **pas nécessaires** à son exécution ;
2. **aidant à le comprendre**.

Des informations supplémentaires ?

1. **commentaires** *dans* le code ;
2. documents externes (*INSTALL* ou *README*) ;
3. *tutoriels d'utilisation* de son code.

À propos (2)

Une simple définition

Ajouter des informations supplémentaires :

1. **pas nécessaires** à son exécution ;
2. **aidant à le comprendre**.

Des informations supplémentaires ?

1. **commentaires** *dans* le code ;
2. documents externes (*INSTALL* ou *README*) ;
3. *tutoriels d'utilisation* de son code.

Quel intérêt ? (1)

Un code plus compréhensible

pour soi pendant qu'on le développe ;

Un code plus compréhensible

pour soi après l'avoir développé ;

Un code plus compréhensible

pour les autres si on veut le distribuer [a].

a. en particulier pour les scripts du Cr@ns donc !

Quel intérêt ? (1)

Un code plus compréhensible

pour soi pendant qu'on le développe ;

Un code plus compréhensible

pour soi après l'avoir développé ;

Un code plus compréhensible

pour les autres si on veut le distribuer [a].

a. en particulier pour les scripts du Cr@ns donc !

Quel intérêt ? (1)

Un code plus compréhensible

pour soi pendant qu'on le développe ;

Un code plus compréhensible

pour soi après l'avoir développé ;

Un code plus compréhensible

pour les autres si on veut le distribuer [a].

a. en particulier pour les scripts du Cr@ns donc !

Quel intérêt ? (2)

Un premier exemple

Pour la fonction `hd` du module `List` de la distribution standard, définie ici :

```
let hd = function
    [] -> failwith "hd"
  | a::l -> a
```

Fichier externe `list.ml` *(lignes 24 à 26).*

Son fonctionnement est mis **dans un commentaire**, dans le fichier `.mli` du module, entre un `(**` et un `*)` :

```
val hd : 'a list -> 'a
(** Return the first element of the given list. Raise
   [Failure "hd"] if the list is empty. *)
```

Fichier externe `list.mli` *(lignes 32 à 34).*

Intérêt d'un outil de documentation

Pas glop !

Lire les commentaires **dans le code** n'est pas pratique...

Permet . . .

- d'intégrer un **outil de recherche** [a].
- de proposer un **index** (listant les modules ou les fonctions) [b].
- d'**intégrer des tests ou des exemples**.
- de **vérifier le code**.

a. *Cf.* http://search.ocaml.jp/?q=hd ou http://docs.python.org/2/search.html?q=socket.socket.

b. *Cf.* http://caml.inria.fr/pub/docs/manual-ocaml/libref/index.html ou http://docs.python.org/2/genindex.html.

Différents formats :

Différents formats : `HTML`, `man`, texte simple (`.txt`), LaTeX (et `PDF`), `TeXinfo`.

Intérêt d'un outil de documentation

Pas glop !

Lire les commentaires **dans le code** n'est pas pratique...

Permet ...

- d'intégrer un **outil de recherche** [a].
- de proposer un **index** (listant les modules ou les fonctions) [b].
- d'**intégrer des tests ou des exemples**.
- de **vérifier le code**.

a. *Cf.* `http://search.ocaml.jp/?q=hd` ou `http://docs.python.org/2/search.html?q=socket.socket`.

b. *Cf.* `http://caml.inria.fr/pub/docs/manual-ocaml/libref/index.html` ou `http://docs.python.org/2/genindex.html`.

Différents formats :

Différents formats : `HTML`, `man`, texte simple (`.txt`), LaTeX (et `PDF`), `TeXinfo`.

Intérêt d'un outil de documentation

Pas glop !

Lire les commentaires **dans le code** n'est pas pratique...

Permet . . .

- d'intégrer un **outil de recherche** [a].
- de proposer un **index** (listant les modules ou les fonctions) [b].
- d'**intégrer des tests ou des exemples**.
- de **vérifier le code**.

a. *Cf.* http://search.ocaml.jp/?q=hd ou http://docs.python.org/2/search.html?q=socket.socket.

b. *Cf.* http://caml.inria.fr/pub/docs/manual-ocaml/libref/index.html ou http://docs.python.org/2/genindex.html.

Différents formats :

Différents formats : `HTML`, `man`, texte simple (`.txt`), LaTeX (et `PDF`), `TeXinfo`.

Pour OCaml

OCamlDoc

OCaml est désormais fourni avec son propre outil de doc : OCamlDoc (depuis la version 3.05).
La documentation officielle d'OCaml est réalisée entièrement avec OCamlDoc depuis juillet 2002.

Pages de références et exemple

La doc pour OCaml 4.00.1 se trouve ici :
http://caml.inria.fr/pub/docs/manual-ocaml-4.00/index.html.
La page de référence pour OCamlDoc est :
http://caml.inria.fr/pub/docs/manual-ocaml-4.00/manual029.html.
La doc officielle ressemble donc à ça : List_official.html#VALhd.

Pour OCaml

OCamlDoc

OCaml est désormais fourni avec son propre outil de doc : OCamlDoc (depuis la version 3.05).
La documentation officielle d'OCaml est réalisée entièrement avec OCamlDoc depuis juillet 2002.

Pages de références et exemple

La doc pour OCaml 4.00.1 se trouve ici :
http://caml.inria.fr/pub/docs/manual-ocaml-4.00/index.html.
La page de référence pour OCamlDoc est :
http://caml.inria.fr/pub/docs/manual-ocaml-4.00/manual029.html.
La doc officielle ressemble donc à ça : List_official.html#VALhd.

Intégrer des commentaires dans le code

Module List

On utilisera le module `List` de la distribution standard pour les exemples (en particulier les deux fonctions `hd` et `tl`) .

```
1  let hd = function
2      [] -> failwith "hd"
3    | a::l -> a
4
5  let tl = function
6      [] -> failwith "tl"
7    | a::l -> l
```

Fichier externe `list.ml` *(lignes 24 à 30).*

```
1  val hd : 'a list -> 'a
2  (** Return the first element of the given list. Raise
3     [Failure "hd"] if the list is empty. *)
4
5  val tl : 'a list -> 'a list
6  (** Return the given list without its first element. Raise
7     [Failure "tl"] if the list is empty. *)
```

Fichier externe `list.mli` *(lignes 32 à 38).*

S'en servir pour générer la documentation (1)

LaTeX, et PDF

```
1 ocamldoc -o List.tex -t List -latex list.mli
2 pdflatex List.tex
```

(à taper dans un terminal)

Produit `List.tex`, et `List.pdf`.

Options :

- `-o List.tex` : fichier de sortie ;
- `-t List` : titre ;
- `-latex` : produit du LaTeX.

S'en servir pour générer la documentation (2)

Fichiers HTML

```
1 ocamldoc -keep-code -colorize-code -d ModuleList ↩
    -t "Module List" -html list.mli
```

(à taper dans un terminal)

Produit `ModuleList/List.html` (qui ne contient aucun détails d'implémentations).

Options :

- `-d ModuleList` : **répertoire** de sortie ;
- `-html` : produit du code HTML.

S'en servir pour générer la documentation (3)

Fichiers HTML avec du code intégré

```
ocamldoc -keep-code -colorize-code -d ↩
    ModuleListCode -t "Module List avec le code" -↩
    html list.ml list.mli
```

(à taper dans un terminal)

Produit `ModuleListCode/List.html` (qui contient le code de chaque élément).

Options :

- `-keep-code` : garde le code (`.ml`) ;
- `-colorize-code` : colore les morceaux de code.

S'en servir pour générer la documentation (4)

Fichier man

```
1 ocamldoc -d "ModuleList_man" -t "Module List" -↩
    man list.mli
```

(à taper dans un terminal)

Produit `ModuleList_man/List.3o`.

Options :

- `-d "ModuleList_man"` : répertoire de sortie ;
- `-man` : génère une page de man.

Pour Python : pyDoc et Sphinx

pyDoc

Python est distribué avec son propre outil de documentation : pyDoc. Aucune documentation de projets conséquents en Python n'est faite avec pyDoc, mais pyDoc est très pratique : c'est lui qui fournit la fonction help !

Sphinx

Sphinx[a] n'est pas intégré à la distribution Python, mais est disponible depuis 2007.
La documentation officielle de Python est désormais réalisée entièrement avec Sphinx.

a. Voir http://sphinx-doc.org/.

Voir la documentation pour Python 2.7.2+ :
http://docs.python.org/2/index.html.

Pour Python : pyDoc et Sphinx

pyDoc

Python est distribué avec son propre outil de documentation : pyDoc. Aucune documentation de projets conséquents en Python n'est faite avec pyDoc, mais pyDoc est très pratique : c'est lui qui fournit la fonction help !

Sphinx

Sphinx[a] n'est pas intégré à la distribution Python, mais est disponible depuis 2007.
La documentation officielle de Python est désormais réalisée entièrement avec Sphinx.

a. Voir http://sphinx-doc.org/.

Voir la documentation pour Python 2.7.2+ :
http://docs.python.org/2/index.html.

Intégrer des commentaires dans le code

Module ANSIColors

Pour les démonstrations suivantes, on utilisera le module ANSIColors [a] de mon projet réseau (en particulier la fonction sprint) .

a. Disponible ici : http://sites.google.com/site/naereencorp/tools/ansi-colors/ANSIColors.py.

En-tête du script :

```
1 #!/usr/bin/env python
2 # -*- encoding: utf-8 -*-
3
4 """ An efficient and simple ANSI colors module (and also a powerfull script), with functions ↩
      to print text using colors.
```

Fichier externe Python2/ANSIColors.py *(lignes 1 à 4).*

Définition d'une fonction :

```
1 def xtitle(title="", verb=False):
2     """ xtitle(title="", verb=False) -> 0|1
3 **Modify the current terminal title**.
4  Returns 0 if one of the two solutions worked, 1 otherwise.
```

Fichier externe Python2/ANSIColors.py *(lignes 421 à 424).*

S'en servir pour depuis Python : la fonction `help`

Appeller PyDoc depuis le l'interpréteur Python

pyDoc est en fait **le module qui permet d'utiliser la fonction `help`**[a]. Ainsi, on peut voir la doc du module `ANSIColors` par la commande suivante :

```
1 cd Python2 && python
2 >>> import ANSIColors
3 >>> help(ANSIColors)
```

(à taper dans un terminal)

a. C'est-à-dire qu'on a `help=pydoc.help`.

La même chose mais dans un `shell` ...

Appeller PyDoc depuis le terminal

Avec `less` :

```
1 pydoc Python2/ANSIColors.py
```

(à taper dans un terminal)

S'en servir pour générer la documentation (2)

Fichiers HTML

```
1 pydoc -w ./
```

(à taper dans un terminal)

Produit Python2/ANSIColors.html#-sprint.

Pour une utilisation plus "simple" de pyDoc, j'ai fait un script Bash : makePydoc.sh.[2]

2. Aussi disponible ici : http://sites.google.com/site/naereencorp/liste-des-projets/makepydoc/makePydoc.sh.

Préliminaire : sphinx-quickstart

```
(mar. févr. 19 -- 18:01:54)<lilian@naereen-corp:[/tmp]> {bashv4.2} 97x27
(18:01:54)lilian@naereen-corp#2[/tmp]$> sphinx-quickstart
Welcome to the Sphinx 1.1.3 quickstart utility.

Please enter values for the following settings (just press Enter to
accept a default value, if one is given in brackets).

Enter the root path for documentation.
> Root path for the documentation [.]:

You have two options for placing the build directory for Sphinx output.
Either, you use a directory "_build" within the root path, or you separate
"source" and "build" directories within the root path.
> Separate source and build directories (y/N) [n]: y

Inside the root directory, two more directories will be created; "_templates"
for custom HTML templates and "_static" for custom stylesheets and other static
files. You can enter another prefix (such as ".") to replace the underscore.
> Name prefix for templates and static dir [_]:

The project name will occur in several places in the built documentation.
> Project name: ANSI Colors
> Author name(s): Lilian Besson
```

Préliminaire (1) : générer un fichier Makefile

Fichier Makefile : avant de commencer

`Makefile` autogénéré par "`sphinx-quickstart`".

```
1  # Makefile for Sphinx documentation
2  #
3
4  # You can set these variables from the command line.
5  SPHINXOPTS    =
6  SPHINXBUILD   = sphinx-build
7  PAPER         =
8  BUILDDIR      = _build
9
10 # Internal variables.
11 PAPEROPT_a4     = -D latex_paper_size=a4
12 PAPEROPT_letter = -D latex_paper_size=letter
13 ALLSPHINXOPTS   = -d $(BUILDDIR)/doctrees $(PAPEROPT_$(PAPER)) $(SPHINXOPTS) .
```

Fichier externe `Python2/Makefile` *(lignes 1 à 13)*.

Préliminaire (2) : générer un fichier conf.py

Fichier conf.py : avant de commencer

conf.py autogénéré par “sphinx-quickstart”.

```
1 # The master toctree document.
2 master_doc = 'index'
3
4 # General information about the project.
5 project = u'ANSI Colors'
6 copyright = u'2013, Lilian Besson'
```

Fichier externe Python2/conf.py *(lignes 39 à 44)*.

Préliminaire (3) : générer un fichier `index.rst`

Des fichiers `.rst` pour paramétriser la doc

Le fichier `Python/index.rst` est important, il fera l'accueil de la doc : on peut y mettre ce qu'on veut.

`.. toctree::` : inclure d'autre modules.

```
1  Contents:
2
3  .. toctree::
4     :maxdepth: 2
5
6     ANSIColors
7     ParseCommandArgs
```

Fichier externe `Python2/index.rst` *(lignes 11 à 17)*.

Préliminaire (4) : écrire des fichier modules.rst

Des fichiers .rst pour paramétriser la doc

Les modules documentés par Sphinx ont besoin d'un fichier .rst, par exemple ANSIColors.rst.[a]

a. J'ai fait un script Bash qui permet d'automatiser ça : pytorst.sh.
Aussi disponible ici : http://sites.google.com/site/naereencorp/liste-des-projets/makepydoc/pytorst.sh.

```
1 ANSIColors Module
2 ==================
3
4 .. automodule:: ANSIColors
5     :members:
6     :private-members:
7     :special-members:
8     :show-inheritance:
```

Fichier externe Python2/ANSIColors.rst *(lignes 1 à 9).*

S'en servir pour générer la documentation (1)

Fichiers HTML

```
1 html:
2   $(SPHINXBUILD) -b html $(ALLSPHINXOPTS) $(BUILDDIR)↩
        /html
3   @echo
4   @echo "Build finished. The HTML pages are in $(↩
        BUILDDIR)/html."
```

Fichier externe `Python/Makefile` *(lignes 153 à 156)*.

Ces lignes produisent notamment `Python2/_build/html/ANSIColors.html#ANSIColors.sprint`, et `Python2/_build/html/_modules/ANSIColors.html#sprint` (qui contient le code de chaque élément).

S'en servir pour générer la documentation (2)

Fichier LaTeX, et PDF

```
1 latexpdf:
2   $(SPHINXBUILD) -b latex $(ALLSPHINXOPTS) $(BUILDDIR↩
        )/latex
3   @echo "Running LaTeX files through pdflatex..."
4   $(MAKE) -C $(BUILDDIR)/latex all-pdf
5   @echo "pdflatex finished; the PDF files are in $(↩
        BUILDDIR)/latex."
```

Fichier externe `Python/Makefile` *(lignes 214 à 218)*.

Ces lignes produisent notamment `Python2/_build/latex/ANSIColors.pdf`.

Mais le rendu n'est pas extra non ?

S'en servir pour générer la documentation (3)

Fichier man

```
1 man:
2     $(SPHINXBUILD) -b man $(ALLSPHINXOPTS) $(BUILDDIR)/↩
          man
3     @echo
4     @echo "Build finished. The manual pages are in $(↩
          BUILDDIR)/man."
```

Fichier externe Python/Makefile *(lignes 225 à 228)*.

Ces lignes produisent notamment Python2/_build/man/ansicolors.1.

S'en servir pour générer la documentation (4)

Fichier texte (.txt)

```
1 text:
2   $(SPHINXBUILD) -b text $(ALLSPHINXOPTS) $(BUILDDIR)↩
        /text
3   @echo
4   @echo "Build finished. The text files are in $(↩
        BUILDDIR)/text."
```

Fichier externe Python/Makefile *(lignes 220 à 223)*.

Ces lignes produisent notamment Python2/_build/txt/ANSIColors.txt.

Démonstration ?

Le format rST

Démonstration *en live*.

- Inclure des modules : `.. toctree::`.
- Inclure une image : `.. image::`.
- Inclure du code : `.. code::`.

Démonstration ?

Le format rST

Démonstration *en live*.

- Inclure des modules : `.. toctree::`.
- Inclure une image : `.. image::`.
- Inclure du code : `.. code::`.

Démonstration ?

Le format rST

Démonstration *en live*.

- Inclure des modules : `.. toctree::`.
- Inclure une image : `.. image::`.
- Inclure du code : `.. code::`.

Pour OCaml

Quelques outils complémentaires

Ces quelques outils peuvent aussi être bien pratiques :

1. `caml2html` permet de convertir des codes `.ml` ou `.mli` en page HTML colorées (voir `list.mli.html`) ;
2. `caml2html` avec l'option **annot** permet de voir le type d'un élément au survol (voir `list.ml.html`).
3. ``ocamlc -i`` permet de générer la signature d'un code `.ml` : une fois votre module terminé, il suffit d'intégrer la doc dans le `.mli`.

Pour OCaml

Quelques outils complémentaires

Ces quelques outils peuvent aussi être bien pratiques :

1. `caml2html` permet de convertir des codes `.ml` ou `.mli` en page HTML colorées (voir `list.mli.html`) ;
2. `caml2html` avec l'option **annot** permet de voir le type d'un élément au survol (voir `list.ml.html`).
3. ``ocamlc -i`` permet de générer la signature d'un code `.ml` : une fois votre module terminé, il suffit d'intégrer la doc dans le `.mli`.

Pour OCaml

Quelques outils complémentaires

Ces quelques outils peuvent aussi être bien pratiques :

1. `caml2html` permet de convertir des codes `.ml` ou `.mli` en page HTML colorées (voir `list.mli.html`) ;
2. `caml2html` avec l'option **annot** permet de voir le type d'un élément au survol (voir `list.ml.html`).
3. ``ocamlc -i'' permet de générer la signature d'un code `.ml` : une fois votre module terminé, il suffit d'intégrer la doc dans le `.mli`.

Pour Python

Quelques outils complémentaires

Ces quelques outils peuvent aussi être bien pratiques :

1. `pyflakes` : analyse statique des erreurs usuelles : assez limité.
2. `pyreverse` : génération de diagramme UML et de diagrammes de dépendances (comme le diagramme ci-dessous) ;
3. `pyhtmlizer` permet de convertir des codes `.py` en page HTML colorées (comme par exemple `Python/ANSIColors.py.html`) ;

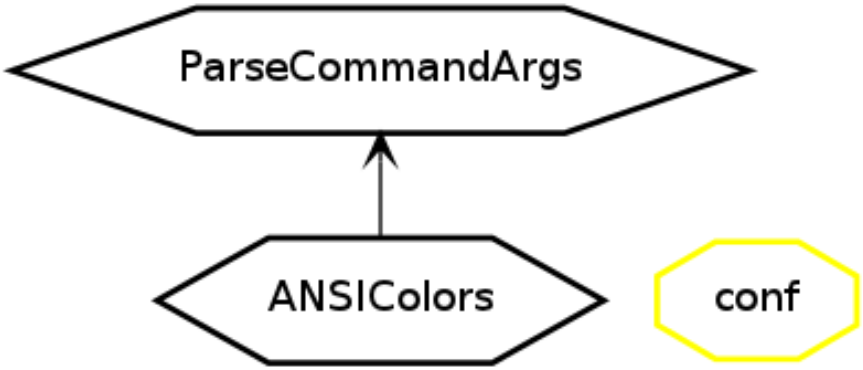

FIGURE: Diagramme liant les différents modules.

Pour Python

Quelques outils complémentaires

Ces quelques outils peuvent aussi être bien pratiques :

1. `pyflakes` : analyse statique des erreurs usuelles : assez limité.
2. `pyreverse` : génération de diagramme `UML` et de diagrammes de dépendances (comme le diagramme ci-dessous) ;
3. `pyhtmlizer` permet de convertir des codes `.py` en page `HTML` colorées (comme par exemple `Python/ANSIColors.py.html`) ;

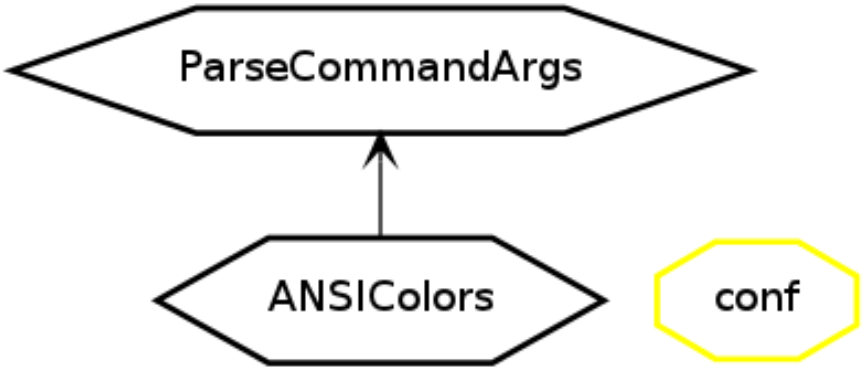

FIGURE: Diagramme liant les différents modules.

Pour Python

Quelques outils complémentaires

Ces quelques outils peuvent aussi être bien pratiques :

1. `pyflakes` : analyse statique des erreurs usuelles : assez limité.
2. `pyreverse` : génération de diagramme UML et de diagrammes de dépendances (comme le diagramme ci-dessous) ;
3. `pyhtmlizer` permet de convertir des codes `.py` en page HTML colorées (comme par exemple `Python/ANSIColors.py.html`) ;

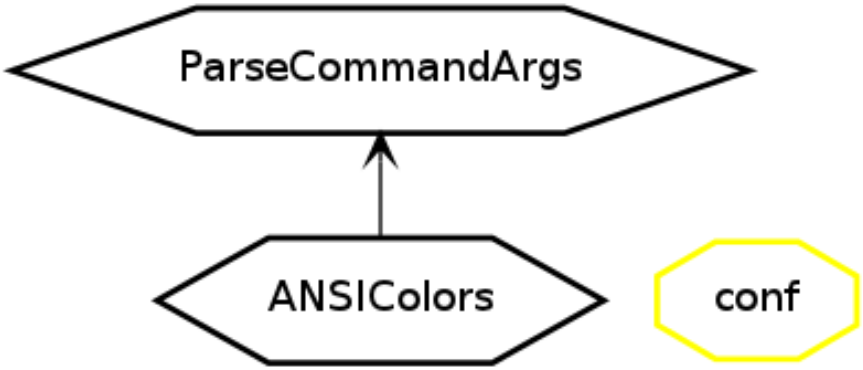

FIGURE: Diagramme liant les différents modules.

Pour Python (2)

Quelques outils complémentaires

Ces quelques outils peuvent aussi être bien pratiques :

1. `pylint` : analyse statique, notation du code, et export des résultats en HTML, texte, ou autres ; Donne une note au code.
2. `pychecker` : anaylyse statique du code, une autre approche : très complet !
3. `pyntch` un genre de typeur statique pour Python. Permet de détecter les erreurs dûes au typage, ou d'**annoter** un fichier source en rajoutant :
   - le type [a] de chaque variable locale et du résultat dans la *docstring* de chaque fonction (ou de méthode),
   - le type de chaque attribut de classe dans la *docstring* de chaque classe,
   - les éventuelles exceptions pouvant être levée dûes au typage [b].

a. Ou dans le cas de plusieurs types possibles, *un type somme*, comme 'int'|'str'.

b. Bien sûr, ces détections ne sont pas parfaites.

Pour Python (2)

Quelques outils complémentaires

Ces quelques outils peuvent aussi être bien pratiques :

1. `pylint` : analyse statique, notation du code, et export des résultats en HTML, texte, ou autres ; Donne une note au code.
2. `pychecker` : anaylyse statique du code, une autre approche : très complet !
3. `pyntch` un genre de typeur statique pour Python. Permet de détecter les erreurs dûes au typage, ou d'**annoter** un fichier source en rajoutant :
   - le type [a] de chaque variable locale et du résultat dans la *docstring* de chaque fonction (ou de méthode),
   - le type de chaque attribut de classe dans la *docstring* de chaque classe,
   - les éventuelles exceptions pouvant être levée dûes au typage [b].

a. Ou dans le cas de plusieurs types possibles, *un type somme*, comme 'int'|'str'.

b. Bien sûr, ces détections ne sont pas parfaites.

Pour Python (2)

Quelques outils complémentaires

Ces quelques outils peuvent aussi être bien pratiques :

1. `pylint` : analyse statique, notation du code, et export des résultats en HTML, texte, ou autres ; Donne une note au code.
2. `pychecker` : anaylyse statique du code, une autre approche : très complet !
3. `pyntch` un genre de typeur statique pour Python. Permet de détecter les erreurs dûes au typage, ou d'**annoter** un fichier source en rajoutant :
   - le type [a] de chaque variable locale et du résultat dans la *docstring* de chaque fonction (ou de méthode),
   - le type de chaque attribut de classe dans la *docstring* de chaque classe,
   - les éventuelles exceptions pouvant être levée dûes au typage [b].

---

a. Ou dans le cas de plusieurs types possibles, *un type somme*, comme 'int'|'str'.

b. Bien sûr, ces détections ne sont pas parfaites.

D'autres outils

On peut enfin citer les autres outils suivants :

- JavaDOC[a] pour Java ;
- Doxygen[b], outil de documentation pour les languages "Orienté Objet" (C++, Java ou même Python[c]) ;
- ack-grep[d] pseudo-clone de grep, conçu spécifiquement pour chercher des motifs dans des morceaux de code ;
- Pygmentize[e] pour colorer n'importe quel fichier source (en HTML ou dans un terminal avec des couleurs ANSI).

---

a. Voir http://docs.oracle.com/javase/6/docs/api/.
b. Voir http://www.stack.nl/~dimitri/doxygen/.
c. Mais pour Python, on lui préfèrera Sphinx, bien plus adapté.
d. Voir http://betterthangrep.com/.
e. Voir http://pygments.org/docs/cmdline/.

D'autres outils

On peut enfin citer les autres outils suivants :

- JavaDOC [a] pour Java ;
- Doxygen [b], outil de documentation pour les languages "Orienté Objet" (C++, Java ou même Python [c]) ;
- ack-grep [d] pseudo-clone de grep, conçu spécifiquement pour chercher des motifs dans des morceaux de code ;
- Pygmentize [e] pour colorer n'importe quel fichier source (en HTML ou dans un terminal avec des couleurs ANSI).

---

a. Voir http://docs.oracle.com/javase/6/docs/api/.
b. Voir http://www.stack.nl/~dimitri/doxygen/.
c. Mais pour Python, on lui préfèrera Sphinx, bien plus adapté.
d. Voir http://betterthangrep.com/.
e. Voir http://pygments.org/docs/cmdline/.

D'autres outils

On peut enfin citer les autres outils suivants :

- JavaDOC [a] pour Java ;
- Doxygen [b], outil de documentation pour les languages "Orienté Objet" (C++, Java ou même Python [c]) ;
- ack-grep [d] pseudo-clone de grep, conçu spécifiquement pour chercher des motifs dans des morceaux de code ;
- Pygmentize [e] pour colorer n'importe quel fichier source (en HTML ou dans un terminal avec des couleurs ANSI).

---

a. Voir http://docs.oracle.com/javase/6/docs/api/.
b. Voir http://www.stack.nl/~dimitri/doxygen/.
c. Mais pour Python, on lui préfèrera Sphinx, bien plus adapté.
d. Voir http://betterthangrep.com/.
e. Voir http://pygments.org/docs/cmdline/.

D'autres outils

On peut enfin citer les autres outils suivants :

- JavaDOC [a] pour Java ;
- Doxygen [b], outil de documentation pour les languages "Orienté Objet" (C++, Java ou même Python [c]) ;
- ack-grep [d] pseudo-clone de grep, conçu spécifiquement pour chercher des motifs dans des morceaux de code ;
- Pygmentize [e] pour colorer n'importe quel fichier source (en HTML ou dans un terminal avec des couleurs ANSI).

---

a. Voir http://docs.oracle.com/javase/6/docs/api/.
b. Voir http://www.stack.nl/~dimitri/doxygen/.
c. Mais pour Python, on lui préfèrera Sphinx, bien plus adapté.
d. Voir http://betterthangrep.com/.
e. Voir http://pygments.org/docs/cmdline/.

Prochain séminaire

À propos

Nous avons parlé plusieurs fois de *colorer du code* ou d'*analyse statique* : Pour plus de détails sur ces deux points là, venez à mon prochain séminaire qui traitera de la coloration syntaxique !

Fin !

Y a-t-il des questions ?

Si oui, n'hésitez pas à les poser, c'est le moment !