

Découverte de l'interpréteur interactif IPython

par Jean PEYROUX (peyroux.developpez.com) Eric
POMMEREAU (eric-pommereau.developpez.com)

Date de publication : 18/01/2008

Dernière mise à jour :

Dans cet article nous vous proposons de partir à la découverte de l'interpréteur interactif IPython. Nous nous efforcerons de vous en faire découvrir le potentiel en étudiant différents cas d'utilisation.

Bonne lecture.

I - Introduction

- I-A - Remerciements
- I-B - Présentation
 - I-B-1 - Historique
 - I-B-2 - Objectifs
 - I-B-3 - Documentation
- I-C - Public visé
- I-D - Installation
 - I-D-1 - Windows
 - I-D-2 - Linux
 - I-D-3 - B.S.D.
 - I-D-4 - MAC OS X

II - Introspection dynamique

- II-A - Complétion
- II-B - Définitions de fonctions
- II-C - Code source d'un module
- II-D - Désignation de l'emplacement du fichier source du module

III - Recherches

- III-A - Variables
- III-B - Méthodes

IV - Interaction avec le shell

- IV-A - Exécution d'une commande shell avec l'opérateur '!'
- IV-B - Capture de la sortie standard d'une commande avec \$ et \$\$
- IV-C - Navigation dans le système de fichiers

V - Les commandes magiques

- V-A - %alias
- V-B - %bookmark
- V-C - %cpaste
- V-D - %dhist
- V-E - %edit
- V-F - %exit, %quit
- V-G - %history ou %hist
- V-H - %macro
- V-I - %lsmagic, %magic
- V-J - %logon, %logoff, %logstart
- V-K - %psearch
- V-L - %psource
- V-M - %quickref
- V-N - %run
- V-O - %save
- V-P - %store
- V-Q - %who, %whos

VI - Utilisation de PDB

VII - Conclusion


I - Introduction

I-A - Remerciements

Nous remercions les personnes qui nous ont aidé à l'amélioration et la relecture de cet article. En particulier **Guigui_**, responsable de la rubrique python qui a pris le temps de tester et de relire notre article.

Nous remercions également Pascal Dauliac pour sa relecture attentive.

I-B - Présentation

Comme IDLE (utilisé par défaut comme interpréteur Python)  **IPython** est un interpréteur Python à la différence qu'il offre des fonctionnalités plus poussées comme **chercher une méthode dans un module** ou un namespace, **afficher le prototype d'une méthode ou d'une fonction**, la complétion dans l'espace de nom local ou bien encore **l'historique des commandes**.

IPython est un véritable outil d'aide au développement ainsi qu'au débogage.

IPython est "cross platform", il est possible de l'utiliser quasiment tout les systèmes d'exploitation.

```
Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)]
Type "copyright", "credits" or "license" for more information.

IPython 0.7.3 -- An enhanced Interactive Python.
?          -> Introduction to IPython's features.
%magic     -> Information about IPython's 'magic' % functions.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

In [1]:
```

La console IPython

Dans l'interpréteur IPython chaque ligne présente le numéro de ligne courant soit en affichant :

```
In [1]:
```

Ou bien, lorsqu'il est utilisé en tant que shell, avec un prompt qui lui aussi présente le numéro de ligne courant :

```
[Console2] |1>
```



IPython est "cross-platform", il peut s'installer sur tout système d'exploitation qui supporte le langage python.

I-B-1 - Historique

IPython est le fruit du regroupement de trois projets :

- IPython de Fernando Pérez
- IPP de Janko Hauser
- LazyPython de Nathan Gray.

Vous pourrez en apprendre davantage sur l'historique d'IPython à l'adresse suivante:

 <http://ipython.scipy.org/doc/manual/node18.html>

I-B-2 - Objectifs

Objectifs d'IPython d'après les concepteurs (traduction du site IPython) :


- **Fournir un interpréteur Python plus puissant** que celui par défaut. IPython propose de nombreuses caractéristiques comme l'introspection d'objet, l'accès au shell système ainsi que ses propres commandes permettant une grande interaction avec l'utilisateur.
- **Proposer un interpréteur embarquable et prêt à l'emploi** pour vos programmes Python. IPython s'efforce d'être un environnement efficace à la fois pour le développement de code Python et pour la résolution des problèmes liés à l'utilisation d'objets Python.
- **Offrir un ensemble de bibliothèques pouvant être utilisé comme environnement** pour d'autres systèmes utilisant Python comme langage sous-jacent (particulièrement pour les environnements scientifiques comme IDLE ou mathematica).
- **Permettre le test interactif des bibliothèques** graphiques gérées comme Tkinter, wxPython, PyGTK alors que IDLE le permet qu'avec des applications Tkinter.

Vous verrez que nos objectifs dans cet article sont bien plus modestes que ceux affichés par les concepteurs.

Nous nous "contenterons" de vous faire découvrir les aspects les plus abordables de ce formidable outil.

I-B-3 - Documentation

Vous trouverez la documentation d'utilisation de IPython à cette adresse:  [Documentation IPython](#).

Il existe également une documentation assez complète sur l'API IPython pour ceux qui veulent intégrer IPython dans leurs codes :  [Documentation IPython A.P.I.](#)

I-C - Public visé


Comme nous l'avons évoqué dans les objectifs d'IPython, ce tutoriel concerne tout public.

Pour les **débutants en Python**, l'utilisation d'IPython ne présente aucun obstacle, bien au contraire, il sera d'une aide précieuse pour la réalisation des premiers scripts.

Pour **les autres** vous avez tout à gagner à passer à IPython, même en complément d'un éditeur spécialisé python.

I-D - Installation


Bien sûr, IPython nécessite l'installation préalable de python.

Vous pouvez télécharger IPython directement sur le site officiel :  **Téléchargement IPython**. Vous y trouverez tout le nécessaire à l'installation quelque soit la plateforme désirée.

I-D-1 - Windows

Pour installer IPython sur la plate-forme MS Windows, vous avez besoin d'installer :

- 1 **Pyreadline** qui apporte une quelques améliorations de la ligne de commande comme la coloration syntaxique ou la complétion.
- 2 **IPython**

Le tout est téléchargeable sur le  **site officiel d'IPython**.

I-D-2 - Linux

```
$ tar -xvzf ipython-0.7.3.tar.gz
$ cd ipython-0.7.3
$ python setup.py build
$ sudo python setup.py install
```

I-D-3 - B.S.D.

Un package est présent sur les environnements BSD. Nous vous invitons à suivre la documentation de votre BSD pour l'installation de ports/package.

I-D-4 - MAC OS X

```
$ tar -zxvf ipython-0.7.3.tar.gz
$ sudo python setup.py install_scripts --install-dir=/usr/local/bin
```

II - Introspection dynamique

L'introspection dynamique consiste à proposer dynamiquement (c'est à dire en fonction du contexte) une liste de choix possibles des différentes propriétés ou méthodes d'un objet.

On obtient ce comportement de l'interpréteur en saisissant le début de l'objet que l'on souhaite utiliser avec la touche [TAB]

II-A - Complétion

IPython utilise l'implémentation de la complétion de readline.

Grâce à IPython, vous pouvez utiliser la complétion sur :

- **Mots clé** par exemple import, def, class
- **Méthodes**, par exemple `os.path.abspath()`
- **Variables**
- Ou bien sur les **noms de fichier dans le répertoire courant**

Prenons un exemple simple, nous souhaitons récupérer dans une variable le chemin courant dans lequel nous sommes.

D'abord, on doit importer le module `os`.

```
imp<TAB>
```

Taper sur tabulation a pour effet de compléter automatiquement le mot clé **import**.



```
In [1]: import
```

Finissons d'importer le module **OS** :

```
import os
```

Ensuite nous souhaitons mettre le chemin courant dans la variable **repertoireCourant** :

```
repertoireCourant = os.get<TAB>
```

Cette fois, taper sur la touche tabulation a pour effet de **proposer les choix possibles** pour les méthodes et propriétés possibles du module `os` (commençant par `get` dans notre exemple). Une fois la bonne méthode trouvée, `getcwd()` en l'occurrence, nous récupérons le chemin dans la variable **repertoireCourant**.

```
In [2]: repertoireCourant = os.get
os.getcwd os.getcwdu os.getenv os.getpid

In [2]: repertoireCourant = os.getcwd()
```

Enfin, nous voulons afficher le contenu de la variable **repertoireCourant** :

```
print rep<TAB>
```

Cette fois, taper sur la touche tabulations nous permet de récupérer le nom de la variable que l'on recherche sous la forme d'une liste de variables disponibles dans le contexte de notre recherche :

```
In [3]: print rep
repr          repertoireCourant

In [3]: print repertoireCourant
c:\Program Files\Console2
```

II-B - Définitions de fonctions

Etant donné la multitude de classes et par extension de méthodes et de propriétés qui en dépendent, il n'est pas évident de connaître par coeur toutes les définitions.

IPython offre la possibilité de lire rapidement les **docstrings** (documentation interne) d'un module, d'une classe, d'une méthode ou d'une propriété.

Reprenons l'exemple précédent en cherchant la documentation de la méthode **getcwd()**

```
os.getcwd?
```

```
In [4]: os.getcwd?
Type:          builtin_function_or_method
Base Class:    <type 'builtin_function_or_method'>
String Form:   <built-in function getcwd>
Namespace:    Interactive
Docstring:
    getcwd() -> path

Return a string representing the current working directory.
```

Le docstring nous indique que cette méthode retourne le chemin courant sous forme d'une chaîne de caractères.

II-C - Code source d'un module

De même, IPython offre la possibilité d'obtenir très simplement **le code source** d'une méthode.

Par exemple, pour connaître le code source de la méthode **basename()** du module **os** :

```
os.path.basename??
```

```
In [6]: os.path.basename??
Type:      function
Base Class: <type 'function'>
String Form: <function basename at 0x00A13FB0>
Namespace: Interactive
File:      c:\python25\lib\ntpath.py
Definition: os.path.basename(p)
Source:
def basename(p):
    """Returns the final component of a pathname"""
    return split(p)[1]
```

Vous constatez qu'un champ **Source** a été ajouté et qu'il contient le code de la méthode. Cette fonctionnalité ne marche qu'avec des modules écrits en Python. Vous ne pourrez pas voir le source d'un module écrit en C comme par exemple le module **pysvn** (utilisation en python du gestionnaire de version subversion).

Cette fonctionnalité nous permet de comprendre rapidement ce que fait notre méthode et comment elle s'utilise.

II-D - Désignation de l'emplacement du fichier source du module

Quand vous utilisez la fonction `?` ou `??` vous remarquerez la présence d'un champ **File** qui représente le chemin absolu du module interrogé.

Pour reprendre notre exemple de la fonction **os.path.basename()**

```
Type:      function
Base Class: <type 'function'>
String Form: <function basename at 0x00A13FB0>
Namespace: Interactive
File:      c:\python25\lib\ntpath.py
Definition: os.path.basename(p)
```

On peut alors voir que notre module est à l'emplacement suivant : **c:\python25\lib\ntpath.py**.

III - Recherches

III-A - Variables

On peut, grâce à la fonction de recherche, **trouver toute variable instanciée** dans notre session IPython. En voici un exemple avec deux variables créées dans la session courante :

```
nom = 'Peyroux'  
prenom = 'Jean'  
  
*nom?
```

```
In [3]: *nom?  
nom  
prenom
```

L'utilisation des "jokers" (* et ?) pour la recherche est tout à fait habituelle, ici on cherche tout ce qui finit par **nom**.

III-B - Méthodes

On peut également rechercher **une méthode dans un module**.

```
import sys  
sys.p*?
```

Dans ce cas nous cherchons l'ensemble des méthodes et propriétés du module **sys** ce qui donne :

```
In [1]: import sys  
  
In [2]: sys.p*?  
sys.path  
sys.path_hooks  
sys.path_importer_cache  
sys.platform  
sys.prefix
```

Un exemple un peu plus complexe avec le module **os** :

```
import os  
*.p*.get*?
```

```
In [1]: import os

In [2]: *.p*.get*?
os.path.getatime
os.path.getctime
os.path.getmtime
os.path.getsize
```

On obtient toutes les méthodes ou propriétés elles mêmes enfant d'une méthode commençant par 'p' de n'importe quel module.




Pour rechercher dans un module, il faut que celui-ci soit préalablement importé.

IV - Interaction avec le shell

Bien que l'équipe d'IPython ne le considère pas comme un "véritable" shell système, IPython peut néanmoins constituer une alternative pertinente à votre shell.

Pour lancer IPython en mode shell il vous suffit de l'exécuter avec les arguments suivants : **-p pysh** . Ceci exécute le mode "InterpreterExec" de IPython.

 Sous Windows, l'installation de IPython place deux raccourcis dans **Démarrer > Programmes > IPython**.

Le raccourci **pysh** lance IPython en mode shell.

Pour obtenir le même résultat en ligne de commande tapez : **C:\Python25\python.exe C:\Python25\scripts\ipython -p sh**.

L'équipe de IPython ne considère pas le mode InterpreterExec comme shell mais plutôt comme un ajout de fonctionnalités et l'apport d'un prompt customisé pour la navigation dans le système de fichiers (comme dans un shell unix). Par ailleurs le mode **-pysh** permet accès aux opérateurs **\$** et **\$\$** pour la capture de la sortie d'une commande système.

On ne peut pas parler de shell au sens strict, bien des options manquent. Par exemple **[ctrl-z]** suspendra le processus de IPython et non plus celui qui "tourne" dans la console.

IV-A - Exécution d'une commande shell avec l'opérateur '!'

Utilisons la commande système **dir** (pour lister le répertoire 'c:\tmp').

```
In [1]: !dir c:\tmp
```

```
In [1]: !dir c:\tmp
IPython system call: dir c:\tmp
Le volume dans le lecteur C n'a pas de nom.
Le numéro de série du volume est D88D-AC05

Répertoire de c:\tmp

21/11/2007  13:45    <REP>          .
21/11/2007  13:45    <REP>          ..
09/11/2007  20:21    <REP>          garmin
16/11/2007  21:14                572 log.txt
17/08/2007  20:17    <REP>          Medion
03/11/2007  12:07    <REP>          msi
21/11/2007  13:33                98 pdb.py
29/11/2007  17:55                96 pdb_test.py
21/11/2007  13:46                1 910 pyLog.log
20/11/2007  18:21                460 python.log
           5 fichier(s)                3 136 octets
           5 Rép(s)    3 546 251 264 octets libres
```


Cette commande a pour effet de lancer la commande système **dir c:\tmp** et d'afficher la sortie de cette commande dans la console

Si vous voulez "capturer" la sortie de votre commande shell sous forme de séquence (tableau), utilisez alors **!!**. Exemple d'utilisation :

```
In [5]: !!dir c:\tmp
```

```
In [2]: !!dir c:\tmp
IPython system call: dir c:\tmp
Out[2]:
[" Le volume dans le lecteur C n'a pas de nom.",
 ' Le num\x82ro de s\x82rie du volume est D88D-AC05',
 '',
 ' R\x82pertoire de c:\tmp',
 '',
 '21/11/2007 13:45 <REP> .',
 '21/11/2007 13:45 <REP> ..',
 '09/11/2007 20:21 <REP> garmin',
 '16/11/2007 21:14 572 log.txt',
 '17/08/2007 20:17 <REP> Medion',
 '03/11/2007 12:07 <REP> msi',
 '21/11/2007 13:33 98 pdb.py',
 '29/11/2007 17:55 96 pdb_test.py',
 '21/11/2007 13:46 1\xff910 pyLog.log',
 '20/11/2007 18:21 460 python.log',
 ' 5 fichier(s) 3\xff136 octets',
 ' 5 R\x82p(s) 3\xff546\xff202\xff112 octets libres']
```

La commande **!!dir** permet de récupérer la sortie de la commande comme une séquence Python.

 **Attention**, l'affichage du résultat de ce type de commande (opérateur **!!**) peut légèrement différer des copies d'écran.

Il est également possible de récupérer le contenu du retour en utilisant le numéro de la ligne de commande préfixé par un blanc souligné (underline).

Par exemple, nous allons mettre le résultat de notre **!!dir** de la ligne 23 dans une variable :

```
monDir = _5
print monDir
# Affiche le résultat du !!dir c:\tmp
```

On peut aussi récupérer directement le résultat d'une commande système en faisant :

```
maVarDir = !dir c:\tmp
```

Pour afficher le contenu de ce que l'on a récupéré :

```
In [19]: for line in maVarDir:
....:     print line
```

```
.....:
```

Le résultat de ces instructions :

```
In [19]: for line in maVarDir:
.....:     print line
.....:
.....:
Le volume dans le lecteur C n'a pas de nom.
Le numéro de série du volume est D88D-AC05

Répertoire de c:\tmp
21/11/2007  13:45    <REP>          .
21/11/2007  13:45    <REP>          ..
09/11/2007  20:21    <REP>          garmin
16/11/2007  21:14           572 log.txt
17/08/2007  20:17    <REP>          Medion
03/11/2007  12:07    <REP>          msi
21/11/2007  13:33           98 pdb.py
29/11/2007  17:55           96 pdb_test.py
21/11/2007  13:46       1 910 pyLog.log
20/11/2007  18:21           460 python.log
           5 fichier(s)             3 136 octets
           5 Rép(s)             3 546 140 672 octets libres
```

IV-B - Capture de la sortie standard d'une commande avec \$ et \$\$

Lorsque IPython est lancé en mode shell (argument **-p pysh**), il est possible de récupérer le contenu d'une commande système dans une variable préfixée par **\$** ou **\$\$**.

Le symbole **\$** permet la capture de la sortie d'une commande système sous forme de texte :

Un exemple sous linux :

```
$monLs = ls
print monLs
Desktop
Documents
Library
```

A la différence du symbole précédent, **\$\$** récupère la sortie sous la forme d'une séquence :

```
peyroux@lapy[~]|28> $$monLs = ls
print monLs
['Desktop', 'Documents', 'Library', 'Movies', 'Music', 'Pictures', 'Public', 'Sites', 'opt']
```

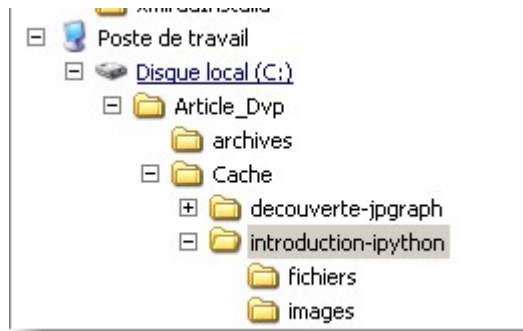
Cela permet de traiter le résultat de votre commande shell comme bon vous semble :

```
# Afficher chaque ligne de la séquence
for fichier in monLs :
|..> print fichier
```

Cette commande affiche sur une ligne chaque entrée de la séquence.

IV-C - Navigation dans le système de fichiers

Partons de l'arborescence suivante :



Navigation dans le système de fichiers

Il est possible de se déplacer facilement dans le système de fichiers en utilisant le début d'un nom de répertoire ou de fichier.

Par exemple avec la commande suivante :

```
%cd c:/Article_Dvp/c[TAB]
```

On obtient les deux répertoires se trouvant dans **c:\article_dvp** qui commencent par la lettre 'c':

```
In [6]: %cd c:/Article_Dvp/c  
c:/Article_Dvp/Cache/ c:/Article_Dvp/css/
```

Idem pour les fichiers où avec la commande suivante :

```
%edit c:/Article_Dvp/Cache/introduction-ipython/i[TAB]
```

On obtient :

```
In [3]: %edit c:/Article_Dvp/Cache/introduction-ipython/i  
c:/Article_Dvp/Cache/introduction-ipython/images  
c:/Article_Dvp/Cache/introduction-ipython/index.php  
c:/Article_Dvp/Cache/introduction-ipython/introduction-ipython.xml
```

V - Les commandes magiques

Les commandes magiques sont un ensemble de commandes spécifiques à IPython précédées de '%'.
Autant dire que ces commandes constituent le "gros morceau" d'IPython.

Vous trouverez dans ce qui suit un inventaire des commandes magiques qui nous ont parues les plus pertinentes dans le cadre de cet article.

V-A - %alias

La commande **%alias** permet de définir un alias pour des commandes système.

Par exemple :

Création de la commande magique %alias

```
# Création de l'alias mon_dir
# Liste des fichiers *.sys du répertoire c:\tmp (récursif).
# Puis lancement de l'alias et affichage du résultat
```

```
In [86]: %alias mon_dir dir "c:\tmp\*.sys" /s /b
```

```
In [87]: mon_dir
IPython system call: dir "c:\tmp\*.sys" /s /b
c:\tmp\Medion\V.1.3.2.5\3xHybrid.sys
c:\tmp\msi\driver\Windows_2000\RT2500.SYS
c:\tmp\msi\driver\Windows_2000\rt2500usb.sys
c:\tmp\msi\driver\Windows_2000\RT61.sys
c:\tmp\msi\driver\Windows_98\RT25009X.SYS
c:\tmp\msi\driver\Windows_98\RT25U98.SYS
c:\tmp\msi\driver\Windows_98\RT619x.sys
c:\tmp\msi\driver\Windows_ME\RT25009X.SYS
c:\tmp\msi\driver\Windows_ME\RT25U98.SYS
c:\tmp\msi\driver\Windows_ME\RT619x.sys
c:\tmp\msi\driver\Windows_XP\RT2500.SYS
c:\tmp\msi\driver\Windows_XP\rt2500usb.sys
c:\tmp\msi\driver\Windows_XP\RT61.sys
c:\tmp\msi\driver\Windows_XP64\rt2500.sys
c:\tmp\msi\driver\Windows_XP64\rt2500usb.sys
c:\tmp\msi\driver\Windows_XP64\rt61.sys
```

Vous pouvez aussi utiliser %s pour afficher les arguments passés lors de l'appel :

```
# Création de l'alias sayHello
# La commande est un echo avec 2 arguments (%s)
# Enfin lancement de la commande par l'alias et affichage du résultat
```

```
In [94]: %alias sayHello echo "Bonjour ton nom est %s et ton prenom est %s"
```

```
In [95]: sayHello Peyroux Jean
IPython system call: echo "Bonjour ton nom est Peyroux et ton prenom est Jean"

"Bonjour ton nom est Peyroux et ton prenom est Jean"
```

V-B - %bookmark

%bookmark vous permet de créer simplement un lien vers un répertoire.

```
# Créer un bookmark du répertoire courant :
%bookmark monBookmark

# Créer un bookmark sur le répertoire '/home/peyroux/Python'
%bookmark maHomePython /home/peyroux/Python

# Lister les bookmarks
%bookmark -l

# Aller dans un répertoire "bookmarké" :
%cd -b maHomePython

# Supprimer un bookmark :
%bookmark -d <name>

# Supprimer tous les bookmark :
%bookmark -r
```

V-C - %cpaste

La commande **%cpaste** permet de coller du code python venant du presse papier afin de l'exécuter.

Une fois tout le code collé, il faut taper '--' pour provoquer l'exécution du contenu.

Un exemple : le code suivant vient d'un fichier texte.

```
monNom = "John Doe"
print monNom + "\n"
print monNom.upper()
```

Une fois copié dans le presse papier, on colle dans la console IPython ces quelques lignes.

```
In [8]: %cpaste
Pasting code; enter '--' alone on the line to stop.

# J'ai fait le coller à cet endroit
:monNom = "John Doe"
:print monNom
:print monNom.upper()
:--

# Le résultat
John Doe
JOHN DOE
```

V-D - %dhist

La commande **%dhist** affiche la liste des répertoires visités durant la session courante

```
# Je vais dans le répertoire 'c:\Documents and Settings'
In [1]: cd c:\Documents and Settings\
c:\Documents and Settings
```

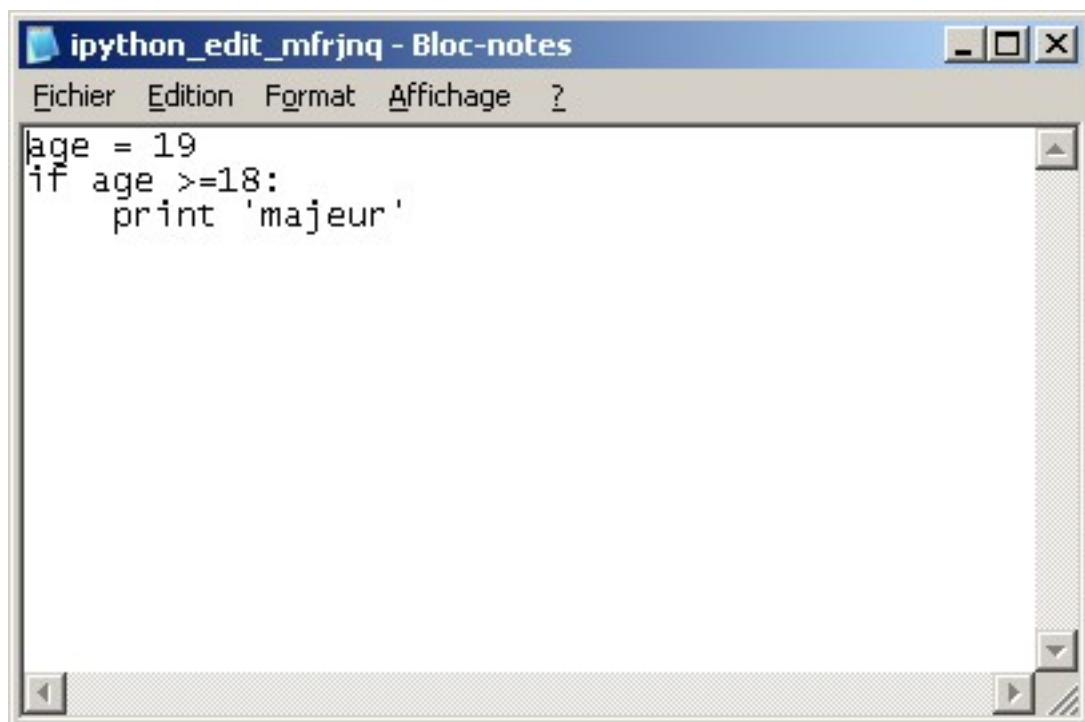


```
In [2]: cd c:\Documents and Settings\eric\Bureau\  
c:\Documents and Settings\eric\Bureau  
  
In [3]: cd  
C:\Documents and Settings\eric  
  
In [4]: %dhist  
Directory history (kept in _dh)  
0: C:\Documents and Settings\eric  
1: c:\Documents and Settings  
2: c:\Documents and Settings\eric\Bureau  
3: C:\Documents and Settings\eric
```

V-E - %edit

Vous pouvez utiliser votre éditeur de texte favori avec IPython pour éditer des instructions tapées. Le code est alors placé dans l'éditeur puis exécuté lors de la fermeture de l'éditeur.

```
In [28]: age = 19  
  
In [29]: if age >=18:  
.....:     print 'majeur'  
.....:  
.....:  
.....:  
majeur  
  
In [30]: %ed 28 29
```



Ouverture de l'éditeur par défaut et ajout du code.

En exécutant **%edit** 2-3 vous allez lancer votre éditeur de texte. Les lignes 2 à 3 sont alors incluses dans l'éditeur et sont exécutées dans IPython après la fermeture.

Vous pouvez aussi utiliser la commande magique de la façon suivante :

```
%edit 2 8 10 21-25
```

Celle-ci éditera les lignes 2, 8 10 puis les lignes situées entre 21 et 25.

Pour ne pas exécuter le code après la fermeture de l'éditeur, utilisez l'argument **-x**

```
# Ouvre l'éditeur et place le code tapé dans IPython des lignes 1 à 4
In [12]: %ed -x 1-4

# Après fermeture le code n'est pas exécuté
```



Il est possible de modifier l'éditeur par défaut (notepad pour Windows) et d'entrer un éditeur personnalisé. Pour cela, il faut éditer le fichier **ipythonrc.ini** qui se trouve dans le répertoire **c:\Documents and Settings\monProfile\ipython** ou dans le répertoire de votre "home" pour les autres systèmes d'exploitation.

V-F - %exit, %quit

Ces deux commandes provoquent la sortie de IPython avec confirmation.

V-G - %history ou %hist

Cette commande permet d'afficher l'historique des commandes tapées dans IPython

Plusieurs utilisations possibles :

- **Sans argument** : retourne les 40 dernières commandes tapées.
- **Un argument** : retourne le nombre de commandes passé en argument.
- **Deux arguments** : retourne les commandes tapées entre le premier et le deuxième argument.

```
# Affichage des 40 dernières lignes
%hist
85 : _ip.system(r'dir "c:\tmp\*.sys" /s /b ')
86 : _ip.magic(r'%alias mondir dir "c:\tmp\*.sys" /s /b')
87 : _ip.system(r'dir "c:\tmp\*.sys" /s /b ')
88 : #?%alias
89 : _ip.magic('%alias sayHello echo "Bonjour %s"')
90 : _ip.system('echo "Bonjour eric" ')
...

# Affichage des 4 dernières lignes :
%hist 4
13: _ip.magic("ls -al")
14: _ip.magic("pwd")
15: _ip.magic("cd $HOME")
16: _ip.magic("mutt")

# Affichage des lignes 3 à 6
%hist 3 6
3: _ip.system("dir /on ")
4: _ip.magic("cd ")
```

```
5: _ip.system("dir /on ")
```

V-H - %macro

%macro permet de définir une portion de code, de le stocker et de l'exécuter ultérieurement.

```
In[1]: nom = "Peyroux"
In[2]: print nom
In[3]: %macro maMacro 1-2
In[4]: maMacro
# Résultat, la macro affiche :
'Peyroux'
```

V-I - %lsmagic, %magic

%lsmagic permet d'obtenir la liste des commandes magiques disponibles.

```
In [22]: %lsmagic
Available magic functions:
%Exit %Pprint %Quit %alias %autocall %autoindent %automagic %bg %bookmar
k %cd %clear %color_info %colors %cpaste %debug %dhist %dirs %ed %edit
%env %exit %hist %history %logoff %logon %logstart %logstate %lsmagic
%macro %magic %p %page %pdb %pdef %pdoc %pfile %pinfo %popd %profile
%prun %psearch %psource %pushd %pwd %pycat %quickref %quit %r %rehash
%rehashdir %rehashx %reset %run %runlog %save %sc %store %sx %system_v
erbose %time %timeit %unalias %upgrade %who %who_ls %whos %xmode

Automagic is ON, % prefix NOT needed for magic functions.
```

%magic est le mode d'emploi détaillé de l'ensemble des commandes magiques.

```
IPython's 'magic' functions
=====

The magic function system provides a series of functions which allow you to
control the behavior of IPython itself, plus a lot of system-type
features. All these functions are prefixed with a % character, but parameters
are given without parentheses or quotes.

NOTE: If you have 'automagic' enabled (via the command line option or with the
%automagic function), you don't need to type in the % explicitly. By default,
IPython ships with automagic on, so you should only rarely need the % escape.

Example: typing '%cd mydir' (without the quotes) changes you working directory
to 'mydir', if it exists.

You can define your own magic functions to extend the system. See the supplied
ipythonrc and example-magic.py files for details (in your ipython
configuration directory, typically $HOME/.ipython/).

You can also define your own aliased names for magic functions. In your
ipythonrc file, placing a line like:
---Return to continue, q to quit---
....
```

V-J - %logon, %logoff, %logstart

L'ensemble de ces commandes permet la prise en charge du log de l'historique des instructions tapées dans IPython.

%logstart permet de démarrer le log.

Le log est créé dans le répertoire courant. Par défaut il est automatiquement nommé 'ipython_log.py'.

Il est possible de spécifier autre emplacement et un autre nom pour le fichier de log :

```
%logstart -r 'c:\tmp\log.txt'
```

Plusieurs options sont possibles concernant le contenu du log :

Argument	Signification
-o	Affiche également le résultat des commandes
-r	Enregistrement brut, tout ce qui est tapé
-t	Affiche la date et l'heure devant chaque commande

Par exemple, à la suite des commandes suivantes tapées dans l'interpréteur IPython :

```
In [1]: %logstart -r 'c:\tmp\python.log'
Activating auto-logging. Current session state plus future input saved.
Filename      : c:\tmp\python.log
Mode          : backup
Output logging : False
Raw input log  : True
Timestamping  : False
State         : active

In [2]: print 'bonjour'
bonjour

In [3]: import os

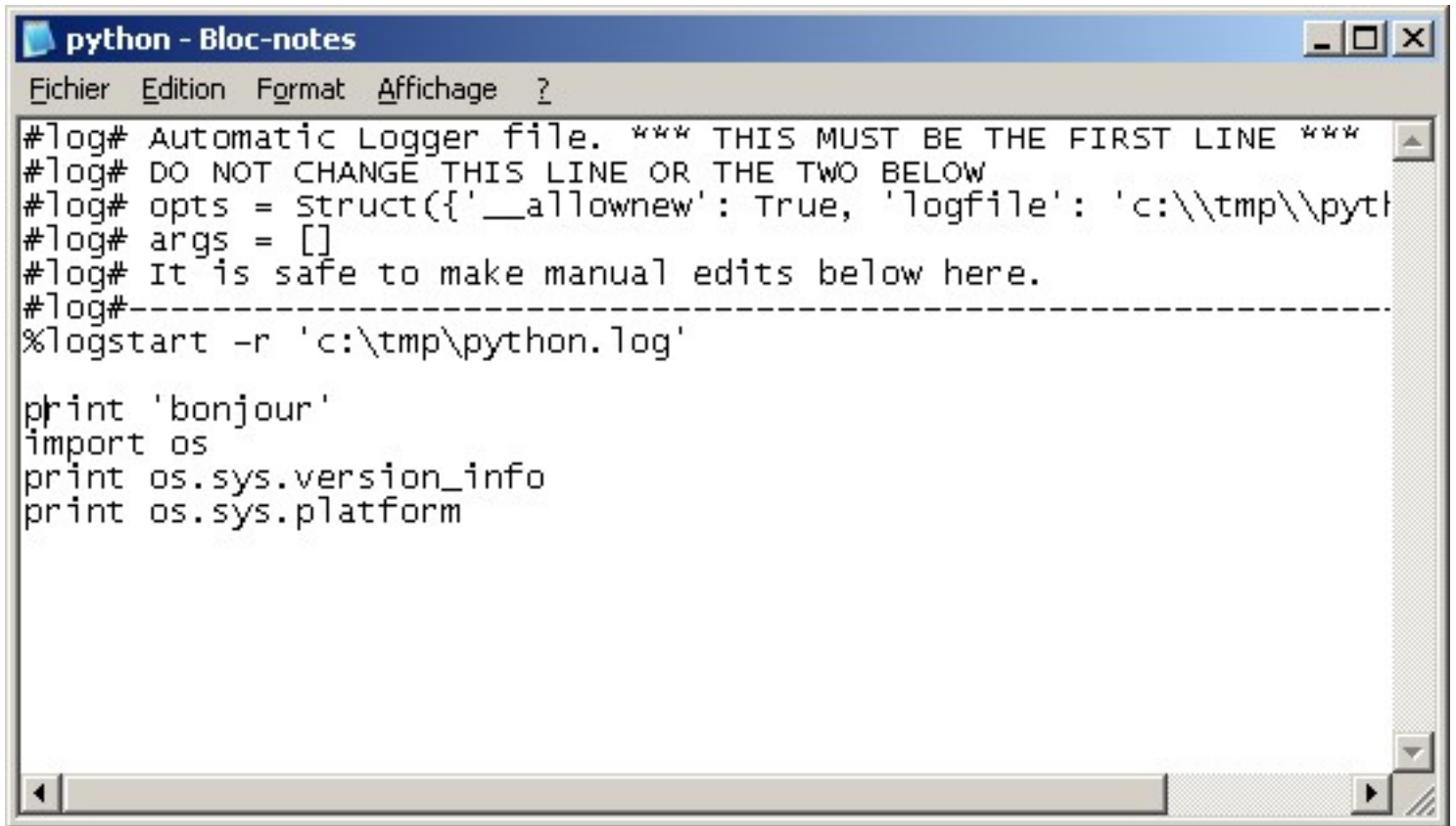
In [4]: print os.sys.platform
win32
```

%logoff interrompt temporairement le log des actions.

%logon redémarre un log suspendu.

Enfin, **%logstate** affiche l'état du log et le fichier dans lequel il est placé.

Si l'on ouvre le fichier 'c:\tmp\python.log', on obtiendra le résultat suivant :



```
python - Bloc-notes
Fichier  Edition  Format  Affichage  ?
#log# Automatic Logger file. *** THIS MUST BE THE FIRST LINE ***
#log# DO NOT CHANGE THIS LINE OR THE TWO BELOW
#log# opts = struct({'__allownew': True, 'logfile': 'c:\\tmp\\pyth
#log# args = []
#log# It is safe to make manual edits below here.
#log#-----
%logstart -r 'c:\\tmp\\python.log'

print 'bonjour'
import os
print os.sys.version_info
print os.sys.platform
```

Contenu du fichier de log

V-K - %psearch

La commande **%psearch** permet la recherche poussée

usage

```
%psearch a* : Objets commençants par un a
%psearch -e builtin a* : Objets qui n'est pas dans le "builtin space" commençant par un a
%psearch a* function : Toutes les fonctions qui commences par un a
%psearch re.e* : Objets qui commencent par un e dans le module re
%psearch r*.e* : Objets qui commencent par un e dans un ou des modules commençant par un r
%psearch r*. * string : Toutes chaines (strings) dans un ou des modules commençant par un r
%psearch -c a* : [-c - Case sensitive] Liste les objets commençants par la minuscule
%psearch -a _* : List des objets commençant par un underscore
```

 **sys.s*?** est équivalent à **%psearch sys.s***

Quelques exemples :

```
# Chercher tout objet contenu dans le module os et commençant par p
In [65]: %psearch os.g*
os.getcwd
os.getcwdu
os.getenv
os.getpid
```

```
# Chercher un module commençant par ft
In [60]: %psearch ft*
ftplib

# Chercher dans le module os une fonction commençant par get
In [63]: %psearch os.get* fonction
os.getenv
```

V-L - %psource

Par exemple, si vous voulez afficher la source de la méthode `login()` du module `ftplib`:

```
import ftplib
In [46]: %psource ftplib.FTP.login
```

Résultat :

```
def login(self, user = '', passwd = '', acct = ''):
    '''Login, default anonymous.'''
    if not user: user = 'anonymous'
    if not passwd: passwd = ''
    if not acct: acct = ''
    if user == 'anonymous' and passwd in ('', '-'):
        # If there is no anonymous ftp password specified
        # then we'll just use anonymous@
        # We don't send any other thing because:
        # - We want to remain anonymous
        # - We want to stop SPAM
        # - We don't want to let ftp sites to discriminate by the user,
        #   host or country.
        passwd = passwd + 'anonymous@'
    resp = self.sendcmd('USER ' + user)
    if resp[0] == '3': resp = self.sendcmd('PASS ' + passwd)
    if resp[0] == '3': resp = self.sendcmd('ACCT ' + acct)
    if resp[0] != '2':
        raise error_reply, resp
    return resp
```

Exemple de recherche dans le module ftplib

V-M - %quickref

La commande `%quickref` affiche la documentation succincte de IPython :

```
IPython -- An enhanced Interactive Python - Quick Reference Card
=====

obj?, obj??, ?obj,??obj    : Get help, or more help for object
?os.p*                    : List names in os starting with p

Example magic:

%alias d ls -F           : 'd' is now an alias for 'ls -F'
alias d ls -F           : Works if 'alias' not a python name
```

```
alist = %alias      : Get list of aliases to 'alist'

System commands:

!cp a.txt b/       : System command escape, calls os.system()
cp a.txt b/        : after %rehashx, most system commands work without !
cp ${f}.txt $bar   : Variable expansion in magics and system commands
files = !ls /usr   : Capture sytem command output
files.s, files.l, files.n: "a b c", ['a','b','c'], 'a\nb\nc'
cd /usr/share      : Obvious, also 'cd d:\home\_ipython' works

History:

_i, _ii, _iii      : Previous, next previous, next next previous input
_i4, _ih[2:5]     : Input history line 4, lines 2-4
exec _i81          : Execute input history line #81 again
_, __, ___        : previous, next previous, next next previous output
_dh                : Directory history
_oh                : Output history
%hist              : Command history

Autocall:

f 1,2              : f(1,2)
/f 1,2             : f(1,2) (forced autoparen)
,f 1 2             : f("1","2")
;f 1 2             : f("1 2")

%Exit:
Exit IPython without confirmation.

%Pprint:
Toggle pretty printing on/off.

%Quit:
Exit IPython without confirmation (like %Exit).

%alias:
Define an alias for a system command.

%autocall:
Make functions callable without having to type parentheses.

%autoindent:
Toggle autoindent on/off (if available).

%automagic:
Make magic functions callable without having to type the initial %.

%bg:
Run a job in the background, in a separate thread.

%bookmark:
Manage IPython's bookmark system.

%cd:
Change the current working directory.

%clear:
Clear various data (e.g. stored history data)

%color_info:
Toggle color_info.

%colors:
Switch color scheme for prompts, info system and exception handlers.

%cpaste:
Allows you to paste & execute a pre-formatted code block from clipboar

%debug:
Activate the interactive debugger in post-mortem mode.

%dhist:
Print your history of visited directories.

%dirs:
Return the current directory stack.

%ed:
Alias to %edit.

%edit:
Bring up an editor and execute the resulting code.

%env:
List environment variables.
```

```
%exit:
    Exit IPython, confirming if configured to do so.
%hist:
    Alternate name for %history.
%history:
    Print input history (_i<n> variables), with most recent last.
%logoff:
    Temporarily stop logging.
%logon:
    Restart logging.
%logstart:
    Start logging anywhere in a session.
%logstate:
    Print the status of the logging system.
%lsmagic:
    List currently available magic functions.
%macro:
    Define a set of input lines as a macro for future re-execution.
%magic:
    Print information about the magic function system.
%p:
    Just a short alias for Python's 'print'.
%page:
    Pretty print the object and display it through a pager.
%pdb:
    Control the automatic calling of the pdb interactive debugger.
%pdef:
    Print the definition header for any callable object.
%pdoc:
    Print the docstring for an object.
%pfile:
    Print (or run through pager) the file where an object is defined.
%pinfo:
    Provide detailed information about an object.
%popd:
    Change to directory popped off the top of the stack.
%profile:
    Print your currently active IPython profile.
%prun:
    Run a statement through the python code profiler.
%psearch:
    Search for object in namespaces by wildcard.
%psource:
    Print (or run through pager) the source code for an object.
%pushd:
    Place the current dir on stack and change directory.
%pwd:
    Return the current working directory path.
%pycat:
    Show a syntax-highlighted file through a pager.
%quickref:
    Show a quick reference sheet
%quit:
    Exit IPython, confirming if configured to do so (like %exit)
%r:
    Repeat previous input.
%rehash:
    Update the alias table with all entries in $PATH.
%rehashdir:
    Add executables in all specified dirs to alias table
%rehashx:
    Update the alias table with all executable files in $PATH.
%reset:
    Resets the namespace by removing all names defined by the user.
%run:
    Run the named file inside IPython as a program.
%runlog:
    Run files as logs.
```



```
%save:
    Save a set of lines to a given filename.
%sc:
    Shell capture - execute a shell command and capture its output.
%store:
    Lightweight persistence for python variables.
%sx:
    Shell execute - run a shell command and capture its output.
%system_verbose:
    Set verbose printing of system calls.
%time:
    Time execution of a Python statement or expression.
%timeit:
    Time execution of a Python statement or expression
%unalias:
    Remove an alias
%upgrade:
    Upgrade your IPython installation
%who:
    Print all interactive variables, with some minimal formatting.
%who_ls:
    Return a sorted list of all interactive variables.
%whos:
    Like %who, but gives some extra information about each variable.
%xmode:
    Switch modes for the exception handlers.
```

V-N - %run

%run execute un script python.

```
Usage : %run [-n -i -t [-N<N>] -d [-b<N>] -p [profile options]] file [args]
```

Exemple d'utilisation courante des différents arguments :

- **-t** : Pour retourner le temp CPU
- **-d** : Pour lancer le débogage pdb
- **-b 40** : Pour commencer le débogage ligne 40

V-O - %save

%save ressemble en tout point à la commande **%edit** mais au lieu d'éditer des commandes IPython, ces dernières sont sauvegardées dans un fichier.

```
%save monFichier 2 8 10-13
```

Sauvegarde dans **monFichier.py** les lignes 2, 8 et les lignes situées entre 10 et 13.

Par défaut, le fichier est enregistré dans votre home. Il est possible de spécifier un chemin, reprenons l'exemple précédent :

```
In [40]: %save "c:\tmp\monFichier.py" 28 29
.....:
```

```
The following commands were written to file `c:\tmp\monFichier.py` :  
age = 19  
if age >=18:  
    print 'majeur'
```

V-P - %store

%store permet une prise en charge "allégée" de la persistance des variables créés dans l'interpréteur IPython (dans la documentation : Lightweight persistence for python variables).

L'usage :

```
%store          - Show list of all variables and their current values  
%store <var>    - Store the *current* value of the variable to disk  
%store -d <var> - Remove the variable and its value from storage  
%store -z       - Remove all variables from storage  
%store -r       - Refresh all variables from store (delete current vals)  
%store foo >a.txt - Store value of foo to new file a.txt  
%store foo >>a.txt - Append value of foo to file a.txt\
```

Quelques exemples d'utilisation :

```
In[1]: nom = "Jean"  
In[2]: %store nom
```

Après avoir quitté IPython, vous pourrez retrouver votre variable sauvée précédemment :

```
In[1]: print nom affiche "Jean"
```

%store permet également de stocker d'autres types de variables comme des listes, des objets ...

```
In [137]: listeDuPathCourant = !dir c:\tmp\garmin\*. * /b  
IPython system call: dir c:\tmp\garmin\*. * /b  
==  
['016901000370.rgn', '042003600370.rgn', 'UPDATE.TXT', 'Updater.exe']  
  
In [138]: %store listeDuPathCourant  
Stored 'listeDuPathCourant' (SList)
```

Il est également possible d'afficher la liste des éléments stockés en tapant simplement **%store**

```
In [139]: %store  
Stored variables and their in-db values:  
listeDuPathCourant      -> ['016901000370.rgn', '042003600370.rgn', 'UPDA  
TE.T  
monNom                  -> 'Jean Peyroux'
```

```
#efface une variable  
%store -d monNom
```

V-Q - %who, %whos

%who retourne la liste des variables en mémoire pour la session courante et des variables et objets persistants.

```
In [31]: who
listeDuPathCourant      monLs      monNom
```

%whos liste détaillée des variables en mémoire de la session courante. Les types et les valeurs sont spécifiés.

```
In [32]: whos
Variable                Type      Data/Info
-----
listeDuPathCourant     SList    ['016901000370.rgn', '042<...>DATE.TXT', 'Updater.exe']
monLs                   SList    ['']
monNom                  str      Jean Peyroux
```

VI - Utilisation de PDB

La librairie python PDB offre la possibilité de déboguer un script python.

IPython permet une utilisation simple et (présentation et coloration améliorés) de PDB.

Prenons le script python suivant que nous avons nommé **pdb_test.py**. Celui-ci est placé dans 'c:\tmp' :

```
def afficherNom(nom):  
    print nom.upper()  
  
if __name__ == "__main__":  
    afficherNom("Peyroux")
```

Pour lancer le débogage d'un script python on utilise la commande **%run** avec l'argument **-d**.

```
%run -d -b1 c:/tmp/pdb_test.py
```

Cette commande a pour conséquence de lancer le débogueur PDB et de démarrer à la ligne 1 (avec l'argument **-b** suivi du numéro de ligne).

Pour aller à la ligne suivante il faut taper 'n' (pour next). Cela permet de voir ligne par ligne le programme se dérouler.

```
Breakpoint 1 at c:\tmp\pdb_test.py:1  
NOTE: Enter 'c' at the ipdb> prompt to start your script.  
> <string>(1) <module>()  
  
ipdb>n  
> c:\tmp\pdb_test.py(1) <module>()  
1---> 1 def afficherNom(nom):  
      2     print nom.upper()  
      3  
  
ipdb>n  
> c:\tmp\pdb_test.py(4) <module>()  
      3  
----> 4 if __name__ == "__main__":  
      5     afficherNom("Peyroux")  
  
ipdb>n  
> c:\tmp\pdb_test.py(5) <module>()  
      3  
----> 4 if __name__ == "__main__":  
----> 5     afficherNom("Peyroux")  
  
ipdb>n  
PEYROUX  
--Return--  
None  
> c:\tmp\pdb_test.py(5) <module>()  
      3  
----> 4 if __name__ == "__main__":  
----> 5     afficherNom("Peyroux")  
ipdb>
```

Utilisation du module PDB dans IPython

A chaque ligne, une flèche verte indique la ligne de code en cours.

Dans cet exemple le script s'est déroulé normalement.

VII - Conclusion

Une petite astuce pour terminer notre article. Si vous souhaitez **imprimer la documentation interne d'IPython** (ce que vous obtenez en tapant `?`) vous pouvez écrire l'usage dans un fichier de la façon suivante :



```
from IPython import usage
fichier = open('C:/doc.txt', 'w')
fichier.write(usage.__doc__)
fichier.close()
```

Le petit tour que nous avons fait d'IPython dans cet article est loin d'être complet en comparaison des nombreuses fonctionnalités proposées.



Nous espérons avoir atteint notre objectif, à savoir de vous avoir donné envie d'utiliser cet interpréteur interactif python. Espérons également que vous avez eu plaisir à nous lire.

Quelques liens utiles :

Liens externes

-  [Site officiel de IPython](#)
-  [Site officiel de python](#)

Liens developpez.com

-  [Forum python sur developpez.com](#)
-  [Section python de developpez.com](#)
-  [Apprendre python](#)
-  [F.A.Q. BSD](#)

