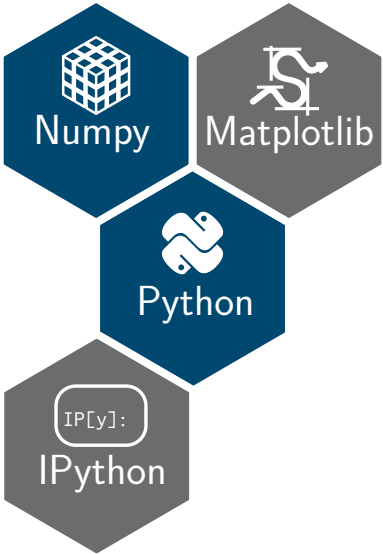

Option « Programmation en Python »

Scripts/modules, programmation orientée objet & exceptions

- **Modularité du code : script/modules**
- **Notion de programmation orientée objet**
- **Gestion des exceptions en Python**
- Entrées/sorties
- Librairie standard : `os`, `sys`, `pickle`

Scripts/modules

- Jusqu'à présent l'ensemble des commandes ou blocs d'instructions ont été tapé et testé directement dans l'interpréteur `ipython`
  - 👍 permet de tester en intéractif le code et sa validité
  - 👎 rend difficile la réutilisation et la modification du code
- Plus la problématique deviendra compliquée, plus le besoin d'écrire du code dans un ou des fichiers, **scripts ou modules**, deviendra pertinente (test, maintenance, lecture du code...)

Scripts/modules

- Jusqu'à présent l'ensemble des commandes ou blocs d'instructions ont été tapé et testé directement dans l'interpréteur `ipython`
  - 👍 permet de tester en intéractif le code et sa validité
  - 👎 rend difficile la réutilisation et la modification du code
- Plus la problématique deviendra compliquée, plus le besoin d'écrire du code dans un ou des fichiers, **scripts ou modules**, deviendra pertinente (test, maintenance, lecture du code...)

- Jusqu'à présent l'ensemble des commandes ou blocs d'instructions ont été tapé et testé directement dans l'interpréteur `ipython`

  👍 permet de tester en intéractif le code et sa validité

  👎 rend difficile la réutilisation et la modification du code

- Plus la problématique deviendra compliquée, plus le besoin d'écrire du code dans un ou des fichiers, **scripts ou modules**, deviendra pertinente (test, maintenance, lecture du code…)

Scripts/modules

- Jusqu'à présent l'ensemble des commandes ou blocs d'instructions ont été tapé et testé directement dans l'interpréteur `ipython`

  👍 permet de tester en intéractif le code et sa validité

  👎 rend difficile la réutilisation et la modification du code

- Plus la problématique deviendra compliquée, plus le besoin d'écrire du code dans un ou des fichiers, **scripts ou modules**, deviendra pertinente (test, maintenance, lecture du code...)

- **Un script** est un fichier contenant un ensemble d'instructions python
- L'extension du fichier-script sera **.py** (indentation, coloration syntaxique…)
- Exemple citation.py

```
1  citation = "Une noisette, j'la casse entre mes fesses tu vois... JCVD"
2  for word in citation.split():
3      print(word)
```

- Le script peut être lancé depuis le terminal *via* la commande

```
$ python citation.py
```

ou directement dans l'interpréteur `ipython` en faisant

```
In [1]: %run citation.py

In [2]: citation
Out[2]: "Une noisette, j'la casse entre mes fesses tu vois... JCVD"
```

- À la différence d'un script, **un module python** est un fichier contenant un **ensemble de fonctions** pouvant être utilisées par différents scripts
- Exemple `jcvd_collection.py`

```python
"""
   A file with a lot of JCVD inside

   This module holds several quotes from Jean-Claude Van Damme
"""

def quote0():
    print("Une noisette, j'la casse entre mes fesses tu vois...")

def quote1():
    print("Quand tu prends confiance en la confiance tu deviens confiant.")

def quote2():
    print("Ce n'est pas moi qui parle...c'est nous qui parlons.")
```

- Pour pouvoir utiliser le module et ses fonctions, **il est nécessaire de l'importer** soit dans un script ou soit dans l'interpréteur
  1. Importation de base

```
In [1]: import jcvd_collection
In [2]: jcvd_collection.quote1()
Quand tu prends confiance en la confiance tu deviens confiant.
```

  2. Importation à l'aide d'un nom raccourci

```
In [1]: import jcvd_collection as jcvd
In [2]: jcvd.quote1()
Quand tu prends confiance en la confiance tu deviens confiant.
```

  3. Importation spécifique d'une fonction

```
In [1]: from jcvd_collection import quote1
In [2]: quote1()
Quand tu prends confiance en la confiance tu deviens confiant.
```

- Pour pouvoir utiliser le module et ses fonctions, **il est nécessaire de l'importer** soit dans un script ou soit dans l'interpréteur
  1. Importation de base

```
In [1]: import jcvd_collection
In [2]: jcvd_collection.quote1()
Quand tu prends confiance en la confiance tu deviens confiant.
```

  2. Importation à l'aide d'un nom raccourci

```
In [1]: import jcvd_collection as jcvd
In [2]: jcvd.quote1()
Quand tu prends confiance en la confiance tu deviens confiant.
```

  3. Importation spécifique d'une fonction

```
In [1]: from jcvd_collection import quote1
In [2]: quote1()
Quand tu prends confiance en la confiance tu deviens confiant.
```

- Pour pouvoir utiliser le module et ses fonctions, **il est nécessaire de l'importer** soit dans un script ou soit dans l'interpréteur

1. Importation de base

```
In [1]: import jcvd_collection
In [2]: jcvd_collection.quote1()
Quand tu prends confiance en la confiance tu deviens confiant.
```

2. Importation à l'aide d'un nom raccourci

```
In [1]: import jcvd_collection as jcvd
In [2]: jcvd.quote1()
Quand tu prends confiance en la confiance tu deviens confiant.
```

3. Importation spécifique d'une fonction

```
In [1]: from jcvd_collection import quote1
In [2]: quote1()
Quand tu prends confiance en la confiance tu deviens confiant.
```

⚠ Lors de l'importation, le module est mis **en cache** et il faut donc le recharger pour que les modifications soient prises en compte

```
In [1]: import importlib
In [2]: importlib.reload(jcvd_collection)
```

- La fonction `help` permet d'accéder à la documentation du module

```
In [1]: help(jcvd_collection)
```

- Par défaut, la localisation des modules se fait dans différents répertoires
  1. dans le répertoire local
  2. dans les répertoires définis au sein de la variable d'environnement `PYTHONPATH`
  3. dans l'ensemble des répertoires référencés par `sys.path`

```
In [1]: import sys
In [2]: sys.path
Out[2]:
['/home/garrido/Developpement/python_d/ipython/dist',
 '/usr/lib/python35.zip',
 '/usr/lib/python3.5',
 '/usr/lib/python3.5/lib-dynload',
 '/home/garrido/Developpement/python_d/ipython/lib/python3.5/site-packages',
 '/home/garrido/Developpement/python_d/ipython/lib/python3.5/site-packages/IPython/extensions',
 '/home/garrido/.ipython']
```

- La fonction `help` permet d'accéder à la documentation du module

```
In [1]: help(jcvd_collection)
```

- Par défaut, la localisation des modules se fait dans différents répertoires
  1. dans le répertoire local
  2. dans les répertoires définis au sein de la variable d'environnement `PYTHONPATH`
  3. dans l'ensemble des répertoires référencés par `sys.path`

```
In [1]: import sys
In [2]: sys.path
['',
 '/home/garrido/Development/python.d/ipython/bin',
 '/usr/lib/python36.zip',
 '/usr/lib/python3.6',
 '/usr/lib/python3.6/lib-dynload',
 '/home/garrido/Development/python.d/ipython/lib/python3.6/site-packages',
 '/home/garrido/Development/python.d/ipython/lib/python3.6/site-packages/IPython/extensions',
 '/home/garrido/.ipython']
```

- La fonction `help` permet d'accéder à la documentation du module

```
In [1]: help(jcvd_collection)
```

- Par défaut, la localisation des modules se fait dans différents répertoires
  1. dans le répertoire local
  2. dans les répertoires définis au sein de la variable d'environnement `PYTHONPATH`
  3. dans l'ensemble des répertoires référencés par `sys.path`

```
In [1]: import sys
In [2]: sys.path
['',
 '/home/garrido/Development/python.d/ipython/bin',
 '/usr/lib/python36.zip',
 '/usr/lib/python3.6',
 '/usr/lib/python3.6/lib-dynload',
 '/home/garrido/Development/python.d/ipython/lib/python3.6/site-packages',
 '/home/garrido/Development/python.d/ipython/lib/python3.6/site-packages/IPython/extensions',
 '/home/garrido/.ipython']
```

- La fonction `help` permet d'accéder à la documentation du module

```
In [1]: help(jcvd_collection)
```

- Par défaut, la localisation des modules se fait dans différents répertoires
  1. dans le répertoire local
  2. dans les répertoires définis au sein de la variable d'environnement `PYTHONPATH`
  3. dans l'ensemble des répertoires référencés par `sys.path`

```
In [1]: import sys
In [2]: sys.path
['',
 '/home/garrido/Development/python.d/ipython/bin',
 '/usr/lib/python36.zip',
 '/usr/lib/python3.6',
 '/usr/lib/python3.6/lib-dynload',
 '/home/garrido/Development/python.d/ipython/lib/python3.6/site-packages',
 '/home/garrido/Development/python.d/ipython/lib/python3.6/site-packages/IPython/extensions',
 '/home/garrido/.ipython']
```

- Il est possible de faire cohabiter au sein d'un même fichier un script et un module

```
1  def quote0():
2      print("Une noisette, j'la casse entre mes fesses tu vois...")
3
4  def quote1():
5      print("Quand tu prends confiance en la confiance tu deviens confiant.")
6
7  def quote2():
8      print("Ce n'est pas moi qui parle...c'est nous qui parlons.")
9
10 # quote0() sera appelé lors du premier import et à chaque exécution
11 quote0()
12
13 if __name__ == "__main__":
14     # quote2() ne sera appelé que lors de l'exécution
15     quote2()
```

Script & modules

```
In [1]: import jcvd_collection
Une noisette, j'la casse entre mes fesses tu vois...

In [2]: import jcvd_collection

In [3]: %run jcvd_collection.py
Une noisette, j'la casse entre mes fesses tu vois...
Ce n'est pas moi qui parle...c'est nous qui parlons.
```

Programmation orientée objet

> *La programmation orientée objet (POO), ou programmation par objet, est* **un paradigme de programmation informatique** *élaboré par les Norvégiens Ole-Johan Dahl et Kristen Nygaard au début des années 1960 et poursuivi par les travaux d'Alan Kay dans les années 1970. Il consiste en la définition et l'interaction de briques logicielles appelées objets ;* **un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre**. *Il possède* **une structure interne et un comportement**, *et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ;* **l'interaction entre les objets via leurs relations** *permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes. Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle.*
>
> *Wikipedia*

> *La programmation orientée objet (POO), ou programmation par objet, est* **un paradigme de programmation informatique** *élaboré par les Norvégiens Ole-Johan Dahl et Kristen Nygaard au début des années 1960 et poursuivi par les travaux d'Alan Kay dans les années 1970. Il consiste en la définition et l'interaction de briques logicielles appelées objets ;* **un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre**. *Il possède* **une structure interne et un comportement**, *et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ;* **l'interaction entre les objets via leurs relations** *permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes. Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle.*
>
> *Wikipedia*

> *La programmation orientée objet (POO), ou programmation par objet, est* **un paradigme de programmation informatique** *élaboré par les Norvégiens Ole-Johan Dahl et Kristen Nygaard au début des années 1960 et poursuivi par les travaux d'Alan Kay dans les années 1970. Il consiste en la définition et l'interaction de briques logicielles appelées objets ;* **un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre**. *Il possède* **une structure interne et un comportement**, *et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ;* **l'interaction entre les objets via leurs relations** *permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes. Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle.*
>
> *Wikipedia*

> *La programmation orientée objet (POO), ou programmation par objet, est* **un paradigme de programmation informatique** *élaboré par les Norvégiens Ole-Johan Dahl et Kristen Nygaard au début des années 1960 et poursuivi par les travaux d'Alan Kay dans les années 1970. Il consiste en la définition et l'interaction de briques logicielles appelées objets ;* **un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre**. *Il possède* **une structure interne et un comportement**, *et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ;* **l'interaction entre les objets via leurs relations** *permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes. Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle.*
>
> *Wikipedia*

> *La programmation orientée objet (POO), ou programmation par objet, est* **un paradigme de programmation informatique** *élaboré par les Norvégiens Ole-Johan Dahl et Kristen Nygaard au début des années 1960 et poursuivi par les travaux d'Alan Kay dans les années 1970. Il consiste en la définition et l'interaction de briques logicielles appelées objets ;* **un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre**. *Il possède* **une structure interne et un comportement**, *et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ;* **l'interaction entre les objets via leurs relations** *permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes. Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle.*
>
> *Wikipedia*

- **Un objet** est une structure hébergeant des **données membres** (ou attributs) et des **fonctions membres** également appelées **méthodes**
- La représentation sous forme d'objet est parfaitement adaptée à la programmation graphique (*GUI*) et à la description des détecteurs en physique (des particules/nucléaire)
- Pour rappel, en Python tout est objet (variables, **fonctions**, classes)

- **Un objet** est une structure hébergeant des **données membres** (ou attributs) et des **fonctions membres** également appelées **méthodes**
- La représentation sous forme d'objet est parfaitement adaptée à la programmation graphique (*GUI*) et à la description des détecteurs en physique (des particules/nucléaire)
- Pour rappel, en Python tout est objet (variables, **fonctions**, classes)

- **Un objet** est une structure hébergeant des **données membres** (ou attributs) et des **fonctions membres** également appelées **méthodes**
- La représentation sous forme d'objet est parfaitement adaptée à la programmation graphique (*GUI*) et à la description des détecteurs en physique (des particules/nucléaire)
- Pour rappel, en Python tout est objet (variables, **fonctions**, classes)

- Déclaration d'un objet/classe `Student`

```
1  class Student:
2      def __init__(self, name):
3          self.name = name
4      def set_age(self, age):
5          self.age = age
6      def set_mark(self, mark):
7          self.mark = mark
```

- **Données membres** : `name`, `age` et `mark`
- **Méthodes** : `__init__`, `set_age`, `set_mark`

- Déclaration d'un objet/classe `Student`

```
1  class Student:
2      def __init__(self, name):
3          self.name = name
4      def set_age(self, age):
5          self.age = age
6      def set_mark(self, mark):
7          self.mark = mark
```

- **Données membres** : `name`, `age` et `mark`
- **Méthodes** : `__init__`, `set_age`, `set_mark`

- Création d'un objet de type `Student`

```
student0 = Student("Patrick Puzo")
student0.set_age(50)
student0.set_mark(0.0)
```

Gestion des exceptions en Python

- Les exceptions sont la conséquence d'erreurs fonctionnelles
  - erreur lors d'un résultat indéfini

```
In [1]: 0/0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-6549dea6d1ae> in <module>()
----> 1 0/0

ZeroDivisionError: division by zero
```

  - erreur typographique dans le nom d'une fonction

```
In [1]: import jcvd_collection
In [2]: quot1()
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-2459ec87cda3> in <module>()
----> 1 quot1()

NameError: name 'quot1' is not defined
```

Gestion des exceptions en Python

- Les exceptions sont la conséquence d'erreurs fonctionnelles
  - erreur lors d'un résultat indéfini

```
In [1]: 0/0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-6549dea6d1ae> in <module>()
----> 1 0/0

ZeroDivisionError: division by zero
```

  - erreur typographique dans le nom d'une fonction

```
In [1]: import jcvd_collection
In [2]: quot1()
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-2459ec87cda3> in <module>()
----> 1 quot1()

NameError: name 'quot1' is not defined
```

Gestion des exceptions en Python

- Les exceptions sont la conséquence d'erreurs fonctionnelles
  - erreur lors d'un résultat indéfini

```
In [1]: 0/0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-6549dea6d1ae> in <module>()
----> 1 0/0

ZeroDivisionError: division by zero
```

  - erreur typographique dans le nom d'une fonction

```
In [1]: import jcvd_collection
In [2]: quot1()
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-2459ec87cda3> in <module>()
----> 1 quot1()

NameError: name 'quot1' is not defined
```

- Pour “attraper” les exceptions avant qu’elles ne causent l’arrêt du programme, on utilise les instructions **try/except**

```
In [1]: while True:
   ...:     try:
   ...:         x = int(input("Veuillez saisir un nombre: "))
   ...:         break
   ...:     except ValueError:
   ...:         print("Je crois avoir demandé un nombre !")
   ...:
Veuillez saisir un nombre: a
Je crois avoir demandé un nombre !
Veuillez saisir un nombre: 11
```

Gestion des exceptions en Python

- Pour “lever” une exception, on utilise l’instruction **`raise`**…

```
In [1]: def achilles_arrow(x):
   ...:     if abs(x - 1) < 1e-3:
   ...:         raise StopIteration
   ...:     x = 1 - (1-x)/2.
   ...:     return x
   ...:
```

- … pour mieux pouvoir la récupérer dans un second bloc

```
In [18]: x = 0
In [19]: while True:
    ...:     try:
    ...:         x = achilles_arrow(x)
    ...:     except StopIteration:
    ...:         break
    ...:

In [20]: x
Out[20]: 0.9990234375
```

- Pour “lever” une exception, on utilise l’instruction **raise**…

```
In [1]: def achilles_arrow(x):
   ...:     if abs(x - 1) < 1e-3:
   ...:         raise StopIteration
   ...:     x = 1 - (1-x)/2.
   ...:     return x
   ...:
```

- … pour mieux pouvoir la récupérer dans un second bloc

```
In [18]: x = 0
In [19]: while True:
    ...:     try:
    ...:         x = achilles_arrow(x)
    ...:     except StopIteration:
    ...:         break
    ...:

In [20]: x
Out[20]: 0.9990234375
```

- La fonction intégrée **print** permet d'afficher à l'écran n'importe quelle chaîne de caractères

```
In [1]: print("Qu'est qu'un chat qui voit dans le futur ?")
```

- La fonction intégrée **input** permet de récupérer une saisie clavier sous la forme d'une chaîne de caractères

```
In [2]: reponse = input("Réponse ? ")
```

- La fonction intégrée **print** permet d'afficher à l'écran n'importe quelle chaîne de caractères

```
In [1]: print("Qu'est qu'un chat qui voit dans le futur ?")
```

- La fonction intégrée **input** permet de récupérer une saisie clavier sous la forme d'une chaîne de caractères

```
In [2]: reponse = input("Réponse ? ")
```

- L'écriture dans un fichier se fait nécessairement par le biais de chaîne de caractères

```
In [1]: f = open("QA.txt", "w")
In [2]: f.write("Qu'est qu'un chat qui voit dans le futur ?")
In [3]: f.close()
```

- La lecture dans un fichier peut se faire de la façon suivante...

```
In [1]: f = open("QA.txt", "r")
In [2]: s = f.read()
In [3]: print(s)
Qu'est qu'un chat qui voit dans le futur ?
In [4]: f.close()
```

- ...ou en lisant le fichier ligne par ligne

```
In [1]: with open("QA.txt", "r") as f:
   ...:     for line in f:
   ...:         print(line)
   ...:
```

L'instruction `with` assure que le fichier sera fermé quoiqu'il advienne notamment si une exception est levée

- La lecture dans un fichier peut se faire de la façon suivante...

```
In [1]: f = open("QA.txt", "r")
In [2]: s = f.read()
In [3]: print(s)
Qu'est qu'un chat qui voit dans le futur ?
In [4]: f.close()
```

- ...ou en lisant le fichier ligne par ligne

```
In [1]: with open("QA.txt", "r") as f:
   ...:     for line in f:
   ...:         print(line)
   ...:
```

L'instruction `with` assure que le fichier sera fermé quoiqu'il advienne notamment si une exception est levée

- Importation du module os

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

- Récupérer le nom du répertoire courant

```
In [1]: os.getcwd()
```

- Lister les fichiers présents dans le répertoire courant

```
In [1]: os.listdir(os.curdir)
```

- Créer un répertoire

```
In [1]: os.mkdir("junkdir")

In [2]: "junkdir" in os.listdir(os.curdir)
Out[2]: True
```

- Renommer et supprimer un répertoire

```
In [1]: os.rename("junkdir", "foodir")

In [2]: os.rmdir("foodir")
In [3]: "foodir" in os.listdir(os.curdir)
Out[3]: False
```

- Supprimer un fichier

```
In [1]: os.remove("junk.txt")
```

👉 Module os : Manipulation des chemins d'accès avec os.path

```
In [1]: %mkdir /tmp/python.d
In [2]: cd /tmp/python.d

In [3]: fp = open("junk.txt", "w"); fp.close()

In [4]: a = os.path.abspath("junk.txt")

In [5]: a
Out[5]: '/tmp/python.d/junk.txt'

In [6]: os.path.split(a)
Out[6]: ('/tmp/python.d', 'junk.txt')

In [7]: os.path.dirname(a)
Out[7]: '/tmp/python.d'

In [8]: os.path.basename(a)
Out[8]: 'junk.txt'

In [9]: os.path.splitext(os.path.basename(a))
Out[9]: ('junk', '.txt')
```

👉 Module os : Manipulation des chemins d'accès avec os.path

```
In [10]: os.path.exists("junk.txt")
Out[10]: True

In [11]: os.path.isfile("junk.txt")
Out[11]: True

In [12]: os.path.isdir("junk.txt")
Out[12]: False

In [13]: os.path.expanduser("~/local")
Out[13]: '/home/jcvd/local'

In [14]: os.path.join(os.path.expanduser("~"), "local", "bin")
Out[14]: '/home/jcvd/local/bin'
```

👉 Module os : Parcourir un répertoire avec os.walk

```
In [1]: for i in range(4):
   ...:     open("junk" + str(i) + ".txt", "w")

In [2]: for dirpath, dirnames, filenames in os.walk(os.curdir):
   ...:     for f in filenames:
   ...:         print(os.path.abspath(f))
/tmp/python.d/junk3.txt
/tmp/python.d/junk2.txt
/tmp/python.d/junk1.txt
/tmp/python.d/junk0.txt
/tmp/python.d/junk.txt

In [3]: import glob
In [4]: for f in glob.glob("*.txt"):
   ...:     os.remove(f)
```

```
In [1]: os.system("ls")
```

⚠ Pour intéragir *via* des commandes systèmes, on priviligiera toutefois le module sh qui, en plus d'être plus complet, fournit des outils pour récupérer le résultat de la commande, les éventuelles erreurs, le code erreur.

Librairie standard

Module sys : Information système

```
In [1]: import sys

In [2]: sys.platform
Out[2]: 'linux'

In [3]: print(sys.version)
3.6.0 (default, Jan 16 2017, 12:12:55)
[GCC 6.3.1 20170109]
```

Librairie standard

👉 Sérialisation d'objets : `pickle`

```
In [1]: import pickle

In [2]: l = [1, None, "Stan"]

In [3]: pickle.dump(l, file("test.pkl", "w"))

In [4]: pickle.load(file("test.pkl"))
Out[4]: [1, None, "Stan"]
```