Python scientifique

Pera Christophe

École Doctorale - Université Lyon 1

26/01/2016

christophe.pera@univ-lyon1.fr
http://flmsn.univ-lyon1.fr
http://www.p2chpd.univ-lyon1.fr
http://lyoncalcul.univ-lyon1.fr

Sources d'informations, non exhaustives !

- Site officiel Python :
  http://python.org
- Python boot camp (orientée calcul scientifique) :
  http://www.pythonbootcamp.info
- Cours en ligne - Mooc (généraliste) :
  http://fr.openclassrooms.com/informatique/python/cours
- Dive Into Python (littérature généraliste) :
  http://www.diveintopython.net/
  http://www.diveintopython3.net/
- tutorial scipy (orienté Python calcul scientifique) :
  http://scipy-lectures.github.io

Objectifs ?

en 4h ...

1. Découvrir le langage Python (Un peu de théorie, beaucoup d'exemples).
2. Découvrir les atouts de Python pour la recherche.
3. Écrire une application en python.
4. Identifier les bonnes pratiques et les limites de son utilisation.
5. Proposer une sélection de ressources pour vous accompagner dans vos projets.

1. Introduction générale au langage et son environnement.
   Interpréteur, programme, syntaxe élémentaire, opérateurs, type de base, fichier, fonctions et gestions d'erreur.
   Représentation des données, modèle objet/classe, liste/tuples/..., strings.
2. Extension Python : Modules et librairies.
   Numpy.
   Scipy.
   Matplotlib.
   Autres.
3. Differences entre les versions 2.x et 3.x de python
4. Mettre en place son environnement python.
5. Études de cas simple, démonstration et optimisation.
6. Allons plus loin ! Accélérons nos calculs en Python.
   Pypy, pythran, cpython, python(x,y), numba (jit), thrust,...
   Comparaisons incomplètes de leur performance.
   High Performance Computing : ipython parallel, multiprocessus, openmp/thread, MPI, opencl, cuda, ...
7. Ressources !

Python, qu'est ce que c'est ...

Résumé

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

https://www.python.org/doc/essays/blurb

Python, qu'est ce que c'est ...

mots clés

Interprété	pas d'étape intermédiaire de compilation.
Orienté objet	concept de programmation basé sur l'objet (une structure de donnée complexe associée à des méthodes/-fonctions).
Haut niveau	indépendant de la manière dont le système interprète et exécute.
Dynamique	possibilité de changer le comportement du code au cours de l'exécution.
Structure de données	moyens de gestions des données.
Script/glue	possibilité de contrôler l'exécution d'autres programmes.
Typage	typage fort et dynamique des variables (int, string, ...).
Syntaxe	grammaire du langage (BNF).
Librairie	collection de code réutilisable.
Binaire	code binaire l'on peut réutiliser/exécuter.

Historique

- Naissance en 1989, par Guido van Rossum (Actuellement : Dropbox).
- Développé dans les année 90s.
- Origine du nom : Monty Python's Flying Circus.
- Développement communautaire "opensource" (license BSD).
- Relies on large community input (bugs, patches) and 3rd party add-on software
- Version 2.0 (2000), 2.6 (2008), 2.7 (2010).
- Version 3.X (2008) . A partir de la version 2.7, le code est facilement "portable" vers une version 3.x.

Python, pourquoi ?

Les alternatives

- C, C++, FORTRAN
  - Pros : excellente performance, "legacy scientific computing" codes.
  - Cons : pas d'interactivité, visualisation difficile, syntaxe complexe, gestion des chaines de charactere compliquée ...
- Mathematica, Maple, Matlab, IDL.
  - Pros : interactif, fonction de visualisation, librairies externes étendues.
  - Cons : couteux, propriétaire, peu "scalable" et inadapté aux taches non-mathématiques.

Python, pourquoi ?

Un langage ouvert et simple.

- Free (BSD licence), multiplateforme (Linux, OSX, Windows, ...).
- Un interpréteur embarqué.
- Une syntaxe lisible et facilement assimilable, proche du C.
- Simple : utilisable par des "non-programmeur".
  - Documentation pléthorique.
  - Gestion automatique de la mémoire.
- Modèle orienté objet complet.
- Type de données riches : lists, sets, dictionaries (hash tables),strings, ...
- Librairie de base.
- Librairies Standards pour IDL/Matlab-like arrays (NumPy).
- "binding" de code existant C, C++ et FORTRAN facile.

Python, pourquoi ?

Richesse des librairies scientifiques.

- Bibliothèques scientifiques :
  - Calcul : NumPy, SciPy, PyIMSL Studio, Sympy, SAGE
  - Système expert : pyCLIPS, pyswip
  - Visualisation : pydot, Matplotlib, pyngl, MayaVi
  - Exploration de données : Orange
  - Simulation : simPy
  - Chimie : PyMOL, MMTK, Chimera, PyQuante
  - Biologie : Biopython
- Analyseur syntaxique : PyParsing
- Graphisme : Pygame, PIL, Soya 3D, Vpython, pymedia, NodeBox
- Framework web : Django, Karrigell, webware, Grok (en), TurboGears, Pylons (en), Flask (en)
- Serveur web, CMS : Plone, Zope, Google App Engine, CherryPy, MoinMoin, django CMS
- Cartographie : TileCache, FeatureServer, Cartoweb 4, Shapely, GeoDjango, PCL

Python, en fin ?

Expérience personnelle ...

- Python n'est pas "très" rapide.
- Certains aspects de Python ont été optimisés (modules performants en C/C++/Fortran comme par exemple scipy/numpy).
- Conseils de programmation :
  - Écrivez votre programme en Python d'abord.
  - Si c'est assez rapide, tant mieux !.
  - Sinon optimisez les parties critiques (et rien d'autre), pas d'optimisation trop précoce.
  - Ca n'est toujours pas assez rapide, penser à paralléliser !
  - Toujours insatisfait ? changer de langage ?

Python, en fin ?

Expérience personnelle ...

Benchmark pour la résolution d'une équation de Laplace 2D sur un modèle 500x500 pour 100 itération.

- Python : 1500.0s
- Python + NumPy : 29.3s
- Matlab : 29.0s
- Weave 2.3, 4.3, 9.5s
- Fortran 77 : 2.9s
- Cython : 2.5s
- C++ : 2.16s

Source 2011 www.scipy.org/PerformancePython.
Remarque : test parallèle : matlab+LKM équivalent à python+numpy+LKM

Premier pas en Python

Comment exécuter des instruction python->Démonstration

- Utiliser un interpréteur Python en ligne de commande.
- Exécuter un fichier de code ou une application python.
- Utiliser ipython en ligne de commande.
- Utiliser ipython notebook.

Répertoire "Demo1" pour les exemples de commandes de test de ces 4 environnements.

Python : la syntaxe

Règle de base

Le langage Python est constitué :

- de mots clefs, qui correspondent à des instructions élémentaires (for , if ...) ;
- de littéraux : valeurs constantes de types variés (25 , 1.e4 , 'abc' ...) ;
- de types intrinsèques (int , float , list , str ...) ;
- d'opérateurs (= , + , * , / ,
- de fonctions intrinsèques (Built-in Functions) qui complètent le langage.

L'utilisateur peut créer :

- des classes : nouveaux types qui s'ajoutent aux types intrinsèques ;
- des objets : entiers, flottants, chaînes, fonctions, programmes, modules... instances de classes définies par l'utilisateur ;
- des expressions combinant identificateurs, opérateurs, fonctions...

Python : la syntaxe

Règle de base

Python est un langage Orienté Objet.

- Avec Python, tout est objet : données, fonctions, modules...
- Un objet :
  - possède une identité (: adresse mémoire) ;
  - possède un type : un objet est l'instanciation d'une classe qui définit son type (type interne : int , float , str ... ou type utilisateur : classxxx) ;
  - contient des données (exple : objet numérique –> sa valeur).

Un objet est référencé par un identificateur :

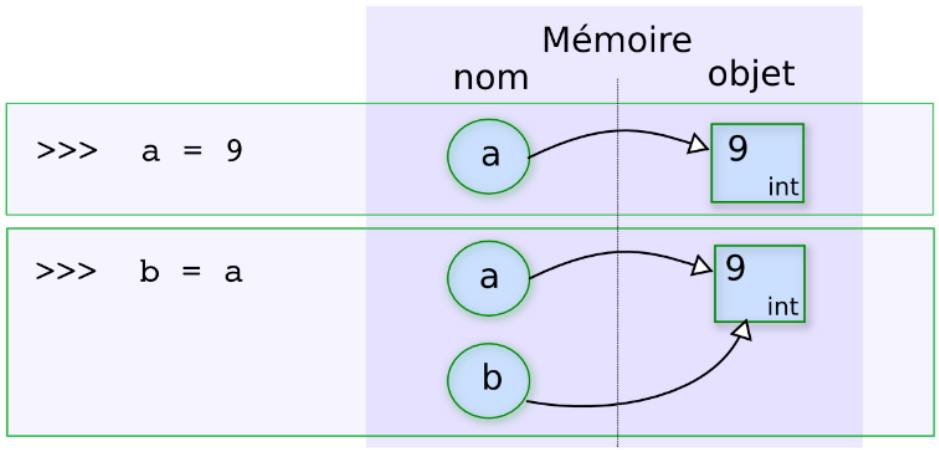
- identificateur, référence, nom, étiquette, sont des termes synonymes ;
- objet et variable sont des termes synonymes (on peut préferer objet) ;
- les noms au format ___xxx___ ont une signification spéciale pour l'interpréteur Python.

Python : la syntaxe

Règle de base : affectation reference -> objet

L'affectation opère de droite à gauche :

- le terme de doite est une expression, qui est évaluée en tant qu'objet ;
- le terme de gauche est l'identificateur (référence, nom, étiquette) affecté à l'objet évalué.
- C'est l'objet qui porte le type et les données (la va-leur, pour un objet numé-rique).

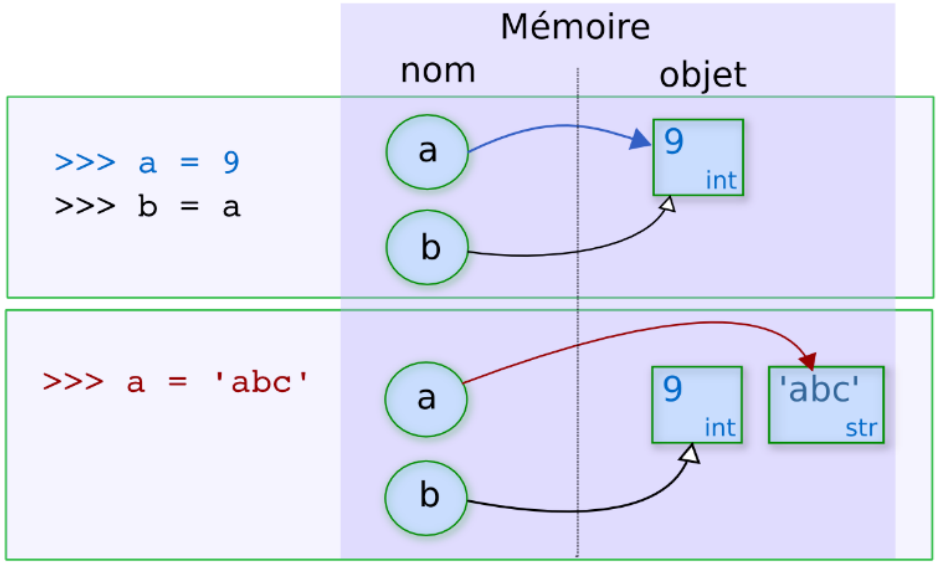


- Un objet ne peut pas changer d'identité (: adresse mémoire), ni de type.
- Un objet peut avoir plusieurs noms (alias).

Python : la syntaxe

Règle de base : affectation reference -> objet

- Un identificateur associé d'abord à un objet peut ensuite référencer un nouvel objet !
- Quand un objet n'a plus de nom (nombre de références nul), il est détruit (mécanisme automatique de "ramasse-miettes", garbage collector).



Python : la syntaxe

Règle de base : les objets

- L'affectation = permet d'affecter un nom à un objet créé ou existant.
- type (objet) renvoie le type de l'objet.
- id (objet) renvoie l'identité de l'objet.
- L'opérateur == teste l'égalité des valeurs de 2 objets.
- L'opérateur is compare l'identité de 2 objets.
- Un identificateur associé d'abord à un objet peut ensuite référencer un nouvel objet !
- Le type d'un objet détermine : ses valeurs possibles, ses opérations supportées, la signification des opérations supportées (opérations polymorphes).

Python : la syntaxe

Règle de base

- L'indentation : Elle différencie les blocs d'instruction :
- Le commentaire :
  - Une ligne commentée commence par un #.
  - Une zone (bloque) commentée commence et se termine par """.

```
If n in range(1,10):
    Print n
Print 'fini'

def myfunction():
    instruction1
    instruction2
```

Python : la syntaxe

La mémoire

- Allocation et des-allocation de mémoire automatique (Garbage collector - références).
- L'assignation du type de la variable est implicite (lors de son utilisation).

```
# Comment allouer une variable
x=1
# une variable reference  un espace memoire
y=x
# 2 variables  peuvent referencer  le  meme espace memoire
x=0
Print y
0
# Reallocation d'une variable, changement du type
y="hello"
```

Python : la syntaxe

Les nombres

- Types de base :
  - Entiers (codé sur 32 bits) : 0 -13 124
  - Entiers longs (précision illimitée) : 1L 340282366920938463463374607431768211456
  - Réels (Float codé sur 64 bits, contient '.' ou 'e' comme indicateur) : 5. 1.3 -4.7 1.23e-6
    Valeurs limites : -1,7x10308 et 1,7x10308
  - Complexes : 1j, -2.5j, 3+4j ; Boooléens : True False
- Opérations :
  - Addition 3+4, 42.+3, 1+0j // soustraction 2-5, 3.-1, 3j-7.5
  - Multiplication 4*3, 2*3.14, 1j*3j
  - Division 1/3, 1./3., 5/3j // power 1.5**3, 2j**2, 2**-0.5
  - Autres opérateurs :

Python : la syntaxe

Les nombres : Attention ...

```
# une comparaison avec un resultat inattendu
0.1 + 0.2 == 0.3
# False

# On farfouille ...
# exemple de formatage de commande print
# afficher les 20 decimales apres la virgule
print("{0:.20f}".format(0.1 + 0.2))
print("{0:.20f}".format(0.3))

#0.30000000000000004441
#0.29999999999999998890
```

Python : la syntaxe

Le typage

- Typage fortement typé, dynamique (Impossible d'additionner un nombre à une chaîne de caractères).
- Pas de 'cast' implicite !
- Des fonctions permettant de transformer les variables dans un autre type.
- Chaque variable devra être déclarée avant d'être utilisée

```
points = 3.2 # points est du type float
print("Tu as " + points + " points !") # Génère une erreur de typage

points = int(points) # points est maintenant du type int (entier), sa valeur est
    arrondie à l'unité inférieure (ici 3)
print("Tu as " + points + " points !") # Génère une erreur de typage

points = str(points) # points est maintenant du type str (chaîne de caractères)
print("Tu as " + points + " points !") # Plus d'erreur de typage, affiche 'Tu as 3
    points !'
```

Python : la syntaxe

Les structures de controle : condition

```
################ principe
#if <condition>:
#   <code>
#elif <condition>:
#   <code>
#else:
#   <code>
################ exemple
test=True
if test:
  print 'vrai'
else:
  print 'faux'

#'vrai'
```

Python : la syntaxe

Les structures de controle : for

```
1  ########## principe
   #for <cible> in <objet>:
3  #  <blocs d'instructions>
   #  if <test1>: break
5  #  if <test2>: continue
   #  else:
7  #    <blocs d'instructions>

9  ########## Exemple
   sum = 0
11 for i in [1, 2, 3, 4]:
     sum += i
13
   prod = 1
15 for p in range(1, 10):
       prod *= p
```

Python : la syntaxe

Les structures de controle : while

```
########## principe
#while <test1>:
#  <blocs d'instructions 1>
#  if <test2>: break
#  if <test3>: continue
#  else:
#    <blocs d'instructions 2>
########## exemple
x = y / 2
while x > 1:
  if y % x == 0:
    print y, 'est facteur de', x
    break
  x = x-1
  else:
print y, 'est premier'
```

Python : la syntaxe

Les fonctions : définition

```
#def <nom_fonction>(arg1, arg2,... argN):
#   ...
#   bloc d'instructions
#   ...
#   return <valeur(s)>

# exemple: Fonction factorielle en Python
def  factorielle (x):
    if  x < 2:
        return 1
    else :
        return x *  factorielle (x-1)
```

Python : la syntaxe

Les fonctions avec plusieurs parametres

```
def table(base, debut=0, fin=11):
  print 'Fragment de la table de multiplication par'\
      , base, ':'
  n = debut
  l = []
  while n < fin :
     print n*base,
     l.append(n*base)
     n += 1
  return l
```

Python : la syntaxe

Les fonctions sans parametres

```
1 # Fonction avec un nombre de parametre variable
  def f(*args, **kwargs):
3   print ,args
    print ,kwargs
5
  f(1, 3, 'b', j = 1)
7 #    (1, 3, 'b')
  #    'j': 1
```

Python : la syntaxe

Les fonctions et les variables

```
# On peut utiliser une variable declarer à l'exterieure d'une fonction
# mais on ne peut pas modifier la valeur de celle-ci
x = 5
def addx(y):
  return x + y
addx(10)
# 15
x
# 5

# Excepté si on utilise le mot clé 'global'
def setx(y):
   global x
   x = y
   print('x is %d' % x)
setx(10)
#x is 10
x
# 10
```

Python : la syntaxe

Les exceptions : traiter une exception

```
### Principe : recuperer une exception
#try:
#  code
#except [exception name] :
#  code
#else:
#  code if no exceptions are raised
#finally:
#  always run on the way out
### Exemple:
def isNumeric(val):
  ### return true if val is a numeric value
  try:
    i = float(val)
  except ValueError, TypeError:
    return False # not numeric
  else:
    return True # numeric
```

Python : la syntaxe

Les exceptions : créer son exception

```
# on peut generer ses propres exceptions:

def laugh(N):
    if N < 0:
        raise ValueError("N must be positive")
    return N * "ha! "

laugh(-4)
# ValueError: N must be positive

laugh(10)
# 'ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! '
```

Python : la syntaxe

Les chaines de charactère

```
"abc" ou 'abc'
'\n'
'abc'+'def' 'abcdef'
3*'abc' 'abcabcabc'
'ab cd e'.split() ['ab','cd','e']
'1,2,3'.split(',') ['1', ' 2', ' 3']
','.join(['1','2']) '1,2'
'a b c '.strip() 'a b c'
'text'.find('ex')     1
'Abc'.upper() 'ABC'
'Abc'.lower() 'abc'
Conversion de int ou float : int('2'), float('2.1')
Conversion de chaines : str(3), str([1, 2, 3])
```

Python : la syntaxe

Les chaines de charactère : concatenation

```
1  # Affectation:
   s = 'i vaut'
3  i = 5

5  # Concaténation: Implicite pour les String, explicite pour les autres types (int,
       reels, bool, ..)

7  print s + " %d %s"%(i, "m.")
   #i vaut 5 m.
9
   print s + ' ' + str(i)
11 #i vaut 5

13 print '*-'*5
   #=>*-*-*-*-*-
```

Python : la syntaxe

Les chaines de charactère : indices

```
print "bonjour"[3], "bonjour"[-1]
#'j' 'r'

print "bonjour"[2:], "bonjour"[:3], "bonjour"[3:5]
#'njour' 'bon' 'jo'

print 'bonjour'[-1::-1]
# 'ruojnob'
```

Python : la syntaxe

Les chaines de charactère : classes et méthodes

- Une classe string : une instance 's'.
- len(s) : renvoie la taille,
- s.find() : recherche une sous-chaîne dans la chaîne,
- s.rstrip() : enlève les espaces de n,
- s.replace() : remplace une chaîne par une autre,
- s.split() : découpe une chaîne,
- s.isdigit() : retourne True si la chaîne contient uniquement des nombres, False sinon,

Python : la syntaxe

Les listes : définitions

- Listes, n-uplets et séquences != tableau langage C (Array)
  => utiliser le module Array (appel biblio C qui céer un tableau C) ou numpy
- Une liste est délimitée par des "[", les n-uplets par des "(".
- Les listes sont "mutables" au contraire des n-uplets.
- len(liste) donne la longueur de la liste.
- list.append() ajoute un élément.
- Les listes et n-uplets sont indexés à partir de 0.
- Pour parcourir ces listes, on utilise la fonction range qui permet d'obtenir une liste d'entiers successifs.
- Les séquences sont des objets dont dérivent les listes, les n-uplets, les chaînes etc....
- Pour afficher les méthodes de la classe "liste" : »> help (list)

Python : la syntaxe

Les listes : exemples

```
# Création >>>
a=[1,2,3,'blabla' ,[9,8]]
# Concaténation >>>
b=[1,2,3]+[4,5]
#  [1,2,3,4,5]

#Ajout d'un élément >>>
a.append('test')
#  [1,2,3,  'blabla  ',[9,8],'  test ']

#Dimension >>>
len(a)
#6

#range([start,] stop[, step]) : Retourne une liste d'entier. >>>
range(5)
# [0,1,2,3,4]

#Une simple indexation >>>
a[0]
#0
```

Python : la syntaxe

Les listes : exemples

```
#Allouer une valeur
a[1]=1
#a = [1, 1, 3, 'blabla', [9, 8], 'test']

#Index négatif:
a [-1]
#'test'
a [-2]
# [9, 8]

#Liste partielle (liste [lower:upper])
a [1:3]
#[1, 3]
a [:3]
#[1, 1, 3]
```

Python : la syntaxe

Les listes : exemples de copie

```
#ATTENTION : distinguer la copie d'une référence et la copie des éléments d'une liste
L = ['Dans', 'python', 'tout', 'est', 'objet']
T = L
T[4] = 'bon'
T
# ['Dans', 'python', 'tout', 'est', 'bon']
L
# ['Dans', 'python', 'tout', 'est', 'bon']
L = T[:]
L[4] = 'objet'
T; L
#['Dans', 'python', 'tout', 'est', 'bon']
#['Dans', 'python', 'tout', 'est', 'objet']
```

Python : la syntaxe

Les listes : classes et méthodes associés

- Une classe Liste : une instance 'L'.
- L.sort() : trier la liste L
- L.append() : ajout d'un élément à la n de la liste L
- L.reverse() : inverser la liste L
- L.index() : rechercher un élément dans la liste L
- L.remove() : retirer un élement de la liste L
- L.pop() : retirer le dernier élément de la liste L ...

Python : la syntaxe

Les dictionnaires : definitions

- Ce sont des "tableaux associatifs"
- Les indices sont appelés "clés".
- d['Nom'] affiche l'élément dont la clé est 'Nom'
- d.keys() affiche les clés
- d.items() affiche les éléments du dictionnaire

Python : la syntaxe

Les dictionnaires : exemples

```
#Création
dico = {}
dico['C'] = 'carbon'; dico['H'] = 'hydrogen'; dico['O'] = 'oxygen'
print dico
# {'C': 'carbone', 'H': 'hydrogen', 'O': 'oxygen'}

#Utilisation
print dico['C']
#carbone

#Création d'un nouveau dictionnaire
dico2={'N':'Nitrogen','Fe':'Iron'}

#Concaténation
dico.update(dico2)
dico
# {'H': 'hydrogen', 'C': 'carbon', 'Fe': 'Iron', 'O': 'oxygn', 'N': 'Nitrogen'}
```

Python : la syntaxe

Les dictionnaires : quelques methodes associés

- len(dico) : taille du dictionnaire
- dico.keys : renvoie les clés du dictionnaire sous forme de liste
- dico.values : renvoie les valeurs du dictionnaire sous forme de liste
- dico.has_key : renvoie True si la clé existe, False sinon
- dico.get : donne la valeur de la clé si elle existe, sinon une valeur par défaut

Python : la syntaxe

Classes et Objets

- Tous les types de base, les fonctions, les instances de classes et les classes elles-mêmes (qui sont des instances de méta-classes) sont des objets.
- Une classe se définit avec le mot class
- Characteristiques : héritage multiple, modification dynamique des attributs, surcharge d'operateur, relexivité/introspection, methodes d'instance/classe/statique.
- Un support limité de l'encapsulation ("we're all consenting adults here")

Python : la syntaxe

Les classes : declaration et utilisation

```
# definition de la classe Student
class Student(object):
    def __init__(self, name):
        self.name = name
    def set_age(self, age):
        self.age = age
    def set_major(self, major):
        self.major = major

anna = Student('anna')
anna.set_age(21)
anna.set_major('physics')
```

Python : la syntaxe

Les classes : héritage

```
# exemple d'heritage / classe Student
class MasterStudent(Student):
  internship = 'mandatory, from March to June'

james = MasterStudent('james')
james.internship
#'mandatory, from March to June'
james.set_age(23)
james.age
#23
```

Python : la syntaxe

Les IOs

```
# ouverture d'un fichier
f = open('inout.dat')
print(f.read())
f.close()
# lecture de ligne dans un fichier
for line in open('inout.dat'):
    print(line.split())
```

Python : la syntaxe

Les IOs

```
# write() is the opposite of read()
contents = open('inout.dat').read()
out = open('my_output.dat', 'w')
out.write(contents.replace(' ', '_'))
out.close()
# writelines() is the opposite of readlines()
lines = open('inout.dat').readlines()
out = open('my_output.dat', 'w')
out.writelines(lines)
out.close()
```

Python : la syntaxe

Les IOs

```
# Don't modify this: it simply writes the example file
f = open('messy_data.dat', 'w')
import random
for i in range(100):
    for j in range(5):
        f.write(' ' * random.randint(0, 6))
        f.write('%0*.*g' % (random.randint(8, 12),
                            random.randint(5, 10),
                            100 * random.random()))
        if j != 4:
            f.write(',')
    f.write('\n')
f.close()
```

Python : extension

Les modules : création

```
# creer son module:
# edition d'un fichier operations.py
def addition(a,b):
    c=a+b
    return c
# Utiliser son module
import operations
operations.addition(1,2)
#3
# Recharger son module
reload(operations)
# Utiliser une partie du module
from operations import addition
addition(1,2)
# Utiliser tous les elements
from operations import *
import numpy
numpy.sin(3.14)
#0.0015926529164868282
## ou
import numpy as N
N.sin(1)
```

Python : extension

Les modules : appel

- Dans le répertoire courant (/homedir/.local/lib/python2.7/site-packages),
- dans PYTHONPATH si défini (même syntaxe que PATH),
- dans un répertoire par défaut (sous Linux : /usr/lib/python),
- syntaxe : from fibo import * (importe tous les noms sauf variables et fonctions privées).

```
1 import sys
  sys.path.append('le chemin de mon module')
3 import mon_module
```

Python : extension

Les modules : standard

- sys : variables système et accès aux options passées en ligne de commande.
- os : informations sur l'OS, manipulations de fichiers et gestions des processus.
  - getcwd() : renvoie le chemin menant au répertoire courant
  - abspath(path) : renvoie le chemin absolu de path
  - exists(path) : renvoie True si path désigne un fichier ou un répertoire existant, False sinon
- math : pi, sqrt, cos, sin, tan,... string time

Python : extension

Les modules : numpy

- Numpy est un ensemble de classes et de fonctions dédiés aux calculs numériques :
  i.d destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux.
  - classe ndarray (N dimensional array) : tableaux homogènes multi-dimensionnels,
  - numpy.linalg : un module d'algèbre linéaire basique,
  - numpy.random : un module pour les générateurs aléatoires,
  - numpy.fft : un module basique de calculs FFT (Fast Fourier Transform).
- Le site web numpy :
  http ://www.numpy.org
- La doc en ligne de numpy :
  http://docs.scipy.org/doc/numpy-1.7.0/reference/

Python : extension

numpy : la classe ndarray

- Structure de donnée élémentaire de type homogène.
- Espaces mémoire contiguës (!= list python).
- Limitations : Opérations non vectorielles ralentisse l'exécution du code (souvent codé en pure python).

Python : extension

numpy : la classe ndarray

```
# creation de tableau
import numpy as np
x=np.array([1, 2, 3])
w = array( [ [1,2], [3,4] ], dtype=complex ) # fixer explicetement le type
### array([1, 2, 3])
y = np.arange(10) # like Python's range, but returns an array
### array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# Operations élémentaires - "terme à terme" !!!
a = np.array([1, 2, 3, 6])
b = np.linspace(0, 2, 4) # create an array with 4 equally spaced points starting with
      0 and ending with 2.
c = a - b
### array([ 1.        , 1.33333333, 1.66666667, 4.        ])
a**2
### array([ 1, 4, 9, 36])
# Fonctions essentiels
a = np.linspace(-np.pi, np.pi, 100)
### array([-3.14159265, -3.07812614, -3.01465962, -2.9511931 , ....
###       ..... 2.9511931 , 3.01465962, 3.07812614, 3.14159265])
b = np.sin(a)
c = np.cos(a)
```

Python : extension

numpy : algebre linéaire

- Algébre linéaire avec numpy et numpy.linalg
  http ://docs.scipy.org/doc/numpy/reference/routines.linalg.html
- Quelques exemples :
  - np.dot multiplication matricielle
  - np.inner produit scalaire
  - np.diag création matrice diagonale
  - np.transpose matrice transposée
  - np.linalg.det déterminant de la matrice
  - np.linalg.inv inverse de la matrice
  - np.linalg.solve résolution du sytème linéaire A x = B
  - np.linalg.eig valeurs propres, vecteurs propres
  - np.linalg.eigvals valeurs propres

Python : extension

numpy : algèbre linéaire

```
import numpy as np
from numpy.random import rand
from numpy.linalg import solve, inv
a = np.array([[1, 2, 3], [3, 4, 6.7], [5, 9.0, 5]])
a.transpose()
#array([[ 1. ,  3. ,  5. ],
#       [ 2. ,  4. ,  9. ],
#       [ 3. ,  6.7,  5. ]])
inv(a)
#array([[-2.27683616,  0.96045198,  0.07909605],
#       [ 1.04519774, -0.56497175,  0.1299435 ],
#       [ 0.39548023,  0.05649718, -0.11299435]])
b = np.array([3, 2, 1])
solve(a, b)  # resomlution de l'equation: ax = b
#array([-4.83050847,  2.13559322,  1.18644068])
c = rand(3, 3)  # création d'une matrice 3x3 aléatoire
c
#array([[ 3.98732789,  2.47702609,  4.71167924],
#       [ 9.24410671,  5.5240412 ,  10.6468792 ],
#       [ 10.38136661,  8.44968437,  15.17639591]])
np.dot(a, c)  # produit matriciel
#array([[ 3.98732789,  2.47702609,  4.71167924],
#       [ 9.24410671,  5.5240412 ,  10.6468792 ],
#       [ 10.38136661,  8.44968437,  15.17639591]])
```

Python : Extension

scipy

- SciPy est une bibliothèque d'algorithme pour les mathématiques, la science, et l'ingénierie.
- Principales classe proposées :
  - Integration (scipy.integrate).
  - Optimization and root finding (scipy.optimize).
  - Interpolation (scipy.interpolate).
  - Fourier Transforms (scipy.fftpack).
  - Signal Processing (scipy.signal).
  - Linear Algebra (scipy.linalg).
  - Compressed Sparse Graph Routines (scipy.sparse.csgraph).
  - Spatial data structures and algorithms (scipy.spatial).
  - Statistics (scipy.stats).
  - Multi-dimensional image processing (scipy.ndimage).
  - Clustering package (scipy.cluster).
  - Orthogonal distance regression (scipy.odr).
  - Sparse matrices (scipy.sparse).
  - Sparse linear algebra (scipy.sparse.linalg).
  - Compressed Sparse Graph Routines (scipy.sparse.csgraph).
  - File IO (scipy.io), Weave C/C++ integration(scipy.weave) ...

Python : Extension

Matplotlib

- Le module matplotlib propose des outils de visualisation : http ://matplotlib.org
- L'aspect graphique proche est proche de Gnunplot ou Matlab.
- Exemples et references : http ://matplotlib.org/gallery.html
  - Tous les graphiques sont accompagnés du programme Python correspondant.
  - Approche par l'exemple (copier/coller/modifier).
- Toute la doc est sur le site http://matplotlib.org/contents.htm.

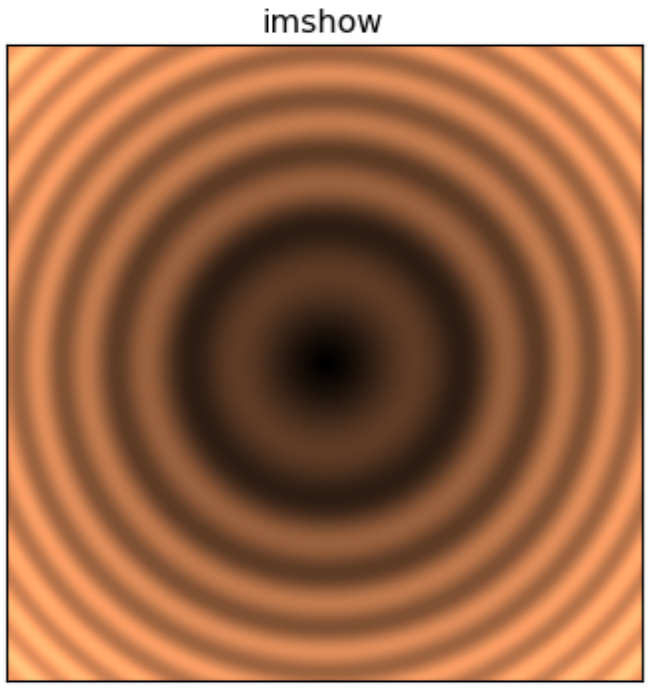
Python : extension

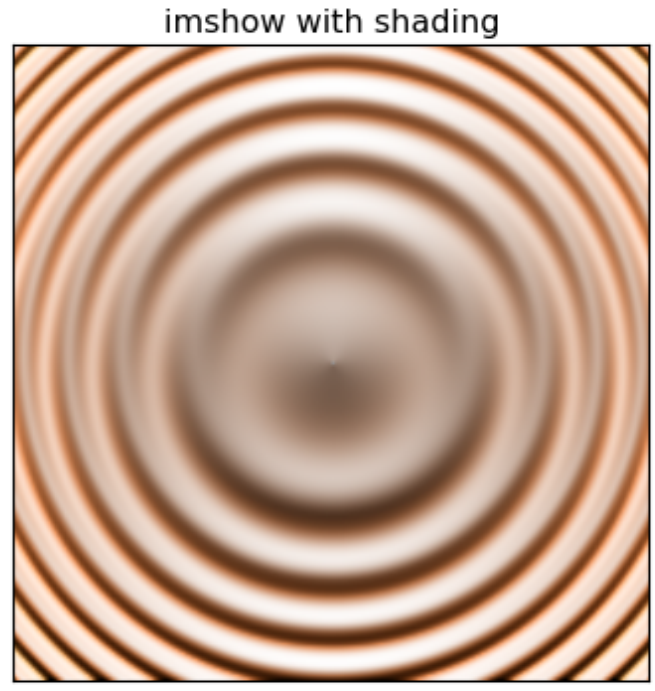
Matplotlib : exemple 2D

```
import numpy as np
import matplotlib.pyplot as  plt
from matplotlib.colors  import LightSource
# example showing how to make shaded relief plots
# like  Mathematica (http://reference.wolfram.com/mathematica/ref/ReliefPlot.html)
# or Generic Mapping Tools (http://gmt.soest.hawaii.edu/gmt/doc/gmt/html/
      GMT_Docs/node145.html)
# test data
X,Y=np.mgrid[-5:5:0.05,-5:5:0.05]
Z=np.sqrt(X**2+Y**2)+np.sin(X**2+Y**2) # create light source object.
ls  = LightSource(azdeg=0,altdeg=65) # shade data, creating an rgb array.
rgb = ls.shade(Z,plt.cm.copper) # plot un-shaded and shaded images.
plt.figure( figsize =(12,5))
plt.subplot(121)
plt.imshow(Z,cmap=plt.cm.copper)
plt. title ('imshow')
plt.xticks ([]) ;  plt.yticks ([])
plt.subplot(122)
plt.imshow(rgb)
plt. title ('imshow with shading')
plt.xticks ([]) ;  plt.yticks ([])
plt.show()
```

Python : extension

Matplotlib : exemple 2D





Python : Extension

matplotlib : exemple 3D

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm

fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
ax.plot_surface(X, Y, Z, rstride=8, cstride=8, alpha=0.3)
cset = ax.contour(X, Y, Z, zdir='z', offset=-100, cmap=cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='x', offset=-40, cmap=cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='y', offset=40, cmap=cm.coolwarm)

ax.set_xlabel('X')
ax.set_xlim(-40, 40)
ax.set_ylabel('Y')
ax.set_ylim(-40, 40)
ax.set_zlabel('Z')
ax.set_zlim(-100, 100)

plt.show()
```

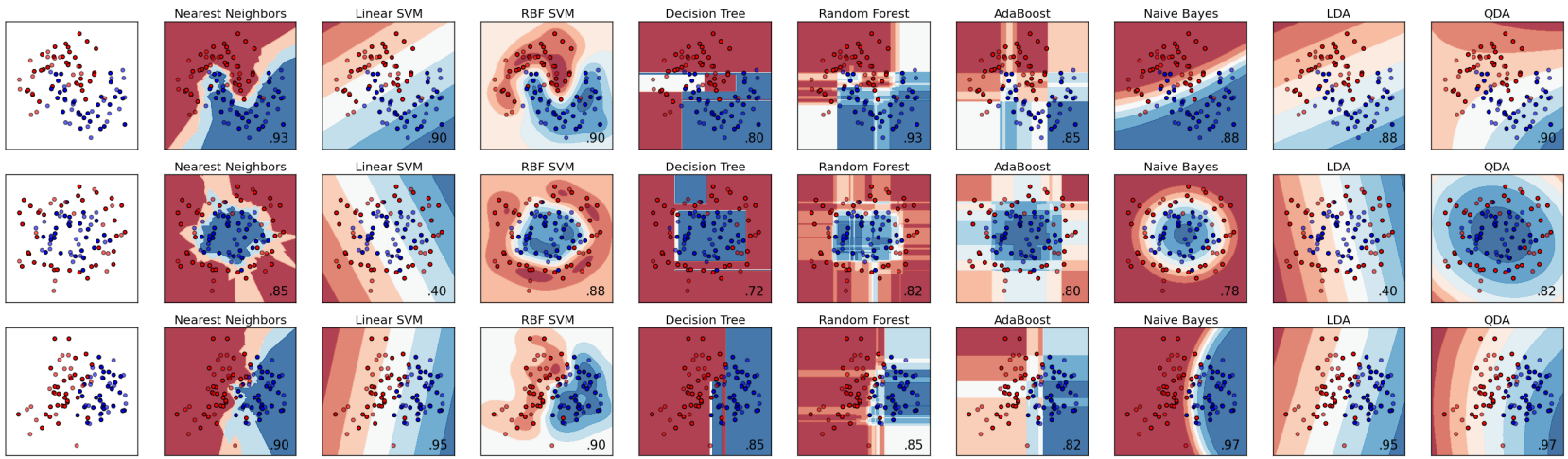
Python : Extension

matplotlib : exemple 3D

Python : Extension

scikit-learn

- Scikit-learn est un module dédié au data mining et à l'analyse de données sous Python.
- Trés dependant de NumPy , Matplotlib et SciPy.
- Exemple des possibilités :
  - Classification - SVM, nearest neighbors, random forest,...
  - Regression - SVR, ridge regression, Lasso, ...
  - Clustering - k-Means, spectral clustering, mean-shift, ...
- http://scikit-learn.org/stable/
- http://scikit-learn.org/stable/auto_examples/index.html



Python : Extension

scikit-learn : exemple SVM (Wikipedia)

- Les machines à vecteurs de support ou séparateurs à vaste marge (en anglais Support Vector Machine, SVM) sont un ensemble de techniques d'apprentissage supervisé destinées à résoudre des problèmes de discriminationnote 1 et de régression. Les SVM sont une généralisation des classifieurs linéaires.
- Les SVM ont été développés dans les années 1990 à partir des considérations théoriques de Vladimir Vapnik sur le développement d'une théorie statistique de l'apprentissage : la théorie de Vapnik-Chervonenkis. Les SVM ont rapidement été adoptés pour leur capacité à travailler avec des données de grandes dimensions, le faible nombre d'hyper paramètres, leurs garanties théoriques, et leurs bons résultats en pratique.
- Les SVM peuvent être utilisés pour résoudre des problèmes de discrimination, décider à quelle classe appartient un échantillon, ou de régression, c'est-à-dire prédire la valeur numérique d'une variable. La résolution de ces deux problèmes passe par la construction d'une fonction h qui à un vecteur d'entrée x fait correspondre une sortie y : y=h(x)

Python : Extension

scikit-learn : non linear SVM 1

```
#Perform binary classification using non-linear SVC with RBF kernel. The target to
      predict is a XOR of the inputs.
#The color map illustrates the decision function learn by the SVC.

print(__doc__)
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
xx, yy = np.meshgrid(np.linspace(-3, 3, 500),
                             np.linspace(-3, 3, 500))
np.random.seed(0)
X = np.random.randn(300, 2)
Y = np.logical_xor(X[:, 0] > 0, X[:, 1] > 0)
# fit the model
clf = svm.NuSVC()
clf.fit(X, Y)
```

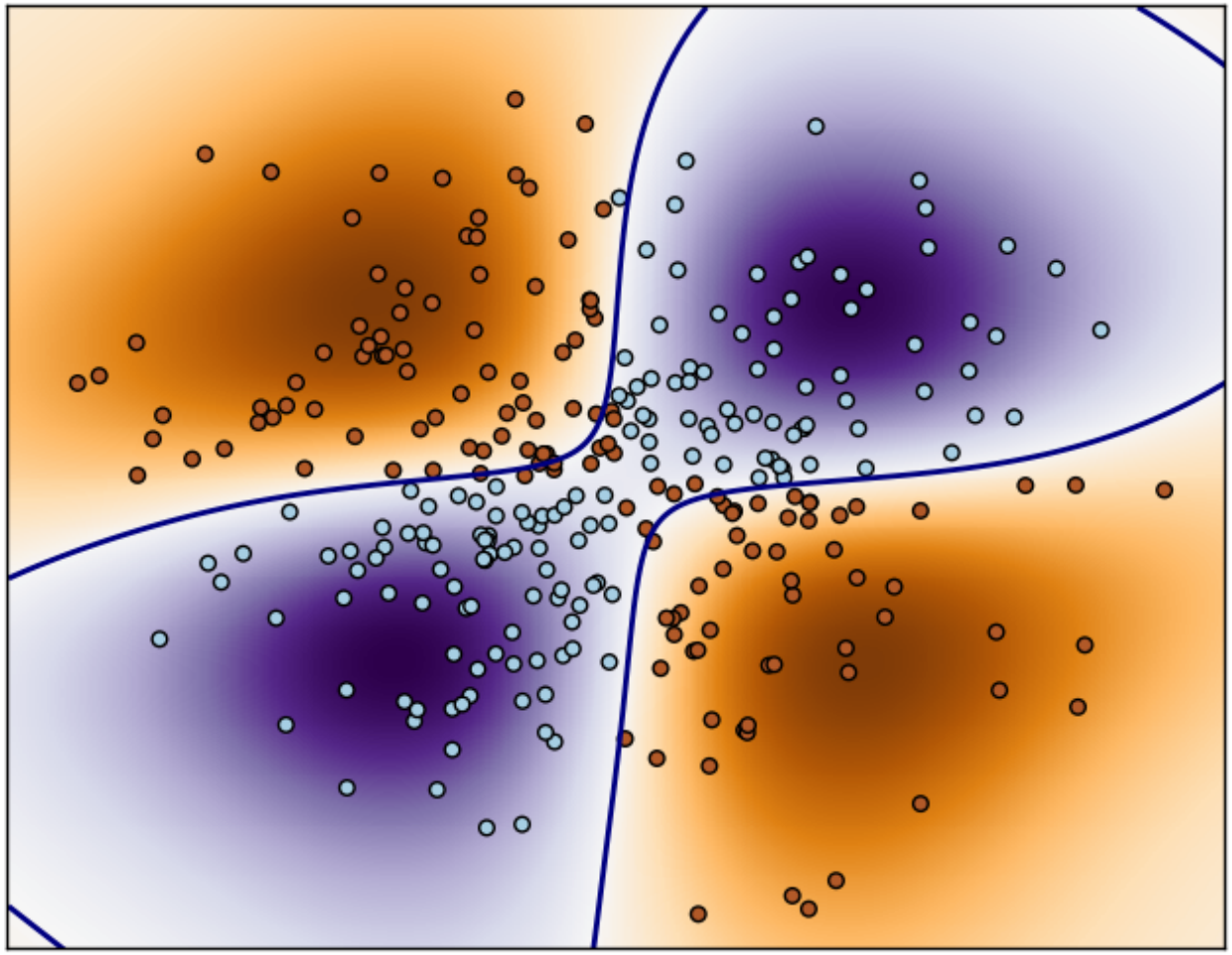
Python : Extension

scikit-learn : non linear SVM 2

```
#Perform binary classification using non-linear SVC with RBF kernel. The target to
    predict is a XOR of the inputs.
#The color map illustrates the decision function learn by the SVC.

# plot the decision function for each datapoint on the grid
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.imshow(Z, interpolation='nearest',
                    extent=(xx.min(), xx.max(), yy.min(), yy.max()), aspect='auto',
                                origin='lower', cmap=plt.cm.PuOr_r)
contours = plt.contour(xx, yy, Z, levels=[0], linewidths=2,
                                 linetypes='--')
plt.scatter(X[:, 0], X[:, 1], s=30, c=Y, cmap=plt.cm.Paired)
plt.xticks(())
plt.yticks(())
plt.axis([-3, 3, -3, 3])
plt.show()
```

Python : Extension

scikit-learn : non linar SVM



Differences entre les evrsion 2.x et 3.x de python

- Python3 : l'instruction print est remplacée par la fonction interne print() .
- Python3 : l'opérateur / effectue toujours la division flottante.
- Python3 : les types int et long sont fusionnés dans le type int . ·
- Python3 : les chaînes de caractères str sont par défaut des chaînes unicode.

```
print "a =", 45.2                    print("a =", 45.2)
# a = 45.2                           # a = 45.2

2/3                                  2/3
#0                                    #0.6666666666666666
2./3                                 2./3
#0.6666666666666666                  #0.6666666666666666
```

Comment installer Python

- Environnement GNU/Linux :
  Installer les paquets/Archives propres à la distribution Linux utilisée :
  Par exemple python2.7, python-numpy, python-scipy,
  python-matplotlib, python-qt4, idle et spyder.
- Environnement Windows :
  Installer la distribution "Python scientifique" Python(x,y), "Python scientifique" Canopy Académique ou Anaconda.
- Environnement Mac OS X :
  Installer la distribution "Python scientifique" Canopy Académique ou Anaconda.
- Distribution scientifique Python x,y :
  http://code.google.com/p/pythonxy/wiki/Downloads
- Distribution Enthought Scineitific Computing Solution :
  https://www.enthought.com/products/canopy/academic/
- Distribution Continuum Analytics :
  http://continuum.io/downloads#all

Exemple d'installation

Distribution Anaconda pour en environnement Linux

- Récupérer le fichier d'installation (script bash = 300Mo) : http://continuum.io/downloads
- Exécuter le script :

Exemple d'installation 2

Distribution Anaconda pour en environnement Linux

```
[xxxx@yyyy python]$ bash Anaconda-2.1.0-Linux-x86_64.sh
Welcome to Anaconda 2.1.0 (by Continuum Analytics, Inc.)
In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
==========================================
...
Do you approve the license terms? [yes|no]
[no] >>> yes
Anaconda will now be installed into this location:
/home/cpera/anaconda
[/home/xxxx/anaconda] >>> /home/xxxx/devel/python/anaconda
PREFIX=/home/xxxx/devel/python/anaconda
installing: python-2.7.8-1 ...
installing: conda-3.7.0-py27_0 ...
...
installing: numba-0.14.0-np19py27_0 ..
installing: numpy-1.9.0-py27_0 ...
installing: matplotlib-1.4.0-np19py27_0 ...
installing: scipy-0.14.0-np19py27_0 ...
...
```

Exemple d'installation 3

Distribution Anaconda pour en environnement Linux

- Tester l'environnement :

```
[xxxx@yyyy python]$ bash Anaconda-2.1.0-Linux-x86_64.sh
...
Do you wish the installer to prepend the Anaconda install location
to PATH in your /home/xxxx/.bashrc ? [yes|no]
[no] >>> yes
Prepending PATH=/home/xxxx/devel/python/anaconda/bin to PATH in /home/
    xxxx/.bashrc
A backup will be made to: /home/xxxx/.bashrc-anaconda.bak
For this change to become active, you have to open a new terminal.
Thank you for installing Anaconda!
[xxxx@yyyyyy python]$ conda list | wc -l
148
[xxxx@yyyyy python]$ du -sh /home/xxxx/devel/python/anaconda/
1,4G    /home/xxxx/devel/python/anaconda/
```

Étude de cas

restons simple ...

- Produit de matrice.
- Mandelbrot.
- lattice Boltzman.

Etude de cas simple

Produit de matrice (wikipedia)

- Si A=(a_{ij}) est une matrice de type (m, n) et B=(b_{ij}) est une matrice de type (n, p), alors leur produit, noté AB=(c_{ij}) est une matrice de type (m, p) donnée par :

$$\bullet c_{ij}=\sum_{k=1}^{n}(a_{ik}+b_{kj})$$

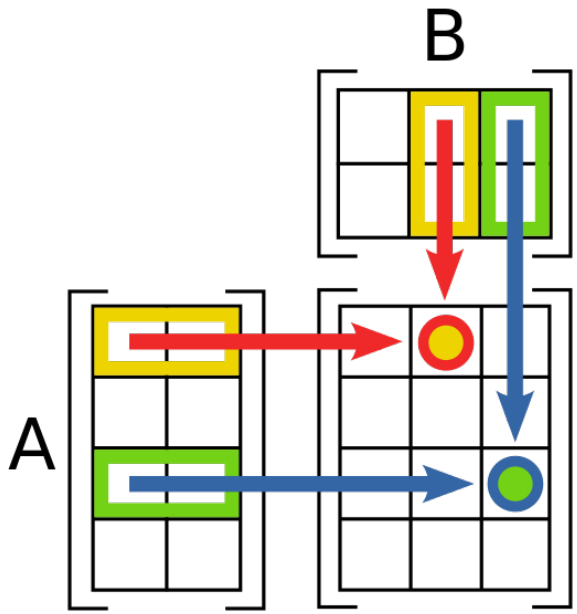


Illustration ...

Etude de cas simple

Produit de matrice

```
import numpy as np

def multiply(m1, m2):
    m = []
    if len(m1[0]) != len(m2):
        print "erreur"
        return False
    for i in range(len(m1)):
        ligne = []
        for j in range(len(m2[0])):
            element = 0
            for k in range(len(m1[0])):
                element = element + m1[i][k] * m2[k][j]
            ligne.append(element)
        m.append(ligne)
    return m

np.random.seed(0)
m1=np.random.random((500,500))
m2=np.random.random((500,500))
m=multiply(m1, m2)
#print m
```

Etude de cas simple

Fractal Mandelbrot (wikipedia)

L'ensemble de Mandelbrot est une fractale définie comme l'ensemble des points c du plan complexe pour lesquels la suite de nombres complexes définie par récurrence par :

$$z_0 = 0 \quad (1)$$

$$z_{n+1} = z_n^2 + c \quad (2)$$

est bornée.

Etude de cas simple

Mandelbrot 1

```
1  import numpy as np
   from pylab import imshow, show
3  from timeit import default_timer as timer

5  #The mandel function performs the Mandelbrot set calculation for a given (x,y)
       position on the imaginary plane. It returns the number of iterations before the
       computation "escapes".
   def mandel(x, y, max_iters):
7  """
   Given the real and imaginary parts of a complex number,
9  determine if it is a candidate for membership in the Mandelbrot
   set given a fixed number of iterations.
11 """
       c = complex(x, y)
13     z = 0.0j
       for i in range(max_iters):
15         z = z*z + c
           if (z.real*z.real + z.imag*z.imag) >= 4:
17             return i
       return max_iters
```

Etude de cas simple

Mandelbrot 2

```
# create_fractal iterates over all the pixels in the image, computing the complex
      coordinates from the pixel coordinates, and calls the mandel function at each
      pixel. The return value of mandel is used to color the pixel.
def create_fractal(min_x, max_x, min_y, max_y, image, iters):
    height = image.shape[0]
    width = image.shape[1]

    pixel_size_x = (max_x - min_x) / width
    pixel_size_y = (max_y - min_y) / height

    for x in range(width):
    real = min_x + x * pixel_size_x
        for y in range(height):
            imag = min_y + y * pixel_size_y
            color = mandel(real, imag, iters)
            image[y, x] = color
# Next we create a 1024x1024 pixel image as a numpy array of bytes. We then call
      create_fractal with appropriate coordinates to fit the whole mandelbrot set.
image = np.zeros((1024, 1536), dtype = np.uint8)
create_fractal (-2.0, 1.0, -1.0, 1.0, image, 20)
imshow(image)
show()
```

benchmark

temps de calcul CPython

- time python mat_mult_basic.py
  - real 1m33.666s
- python mandelbrot.py
  - Mandelbrot created in 8.874437 s

Pourquoi Optimiser

Python est lent

- Typage dynamique.
- Optimisation de code difficile pour un langage interprété.
- Gestion non optimale des acces à la mémoire.

Pourquoi Optimiser

Pseudocode C, c=a+b

```
#/* C code */
#int a = 1;
#int b = 2;
#int c = a + b;

# suite des operations executé par le  binaire
Assign <int> 1 to a
Assign <int> 2 to b
call  binary_add<int, int>(a, b)
Assign the result  to c
```

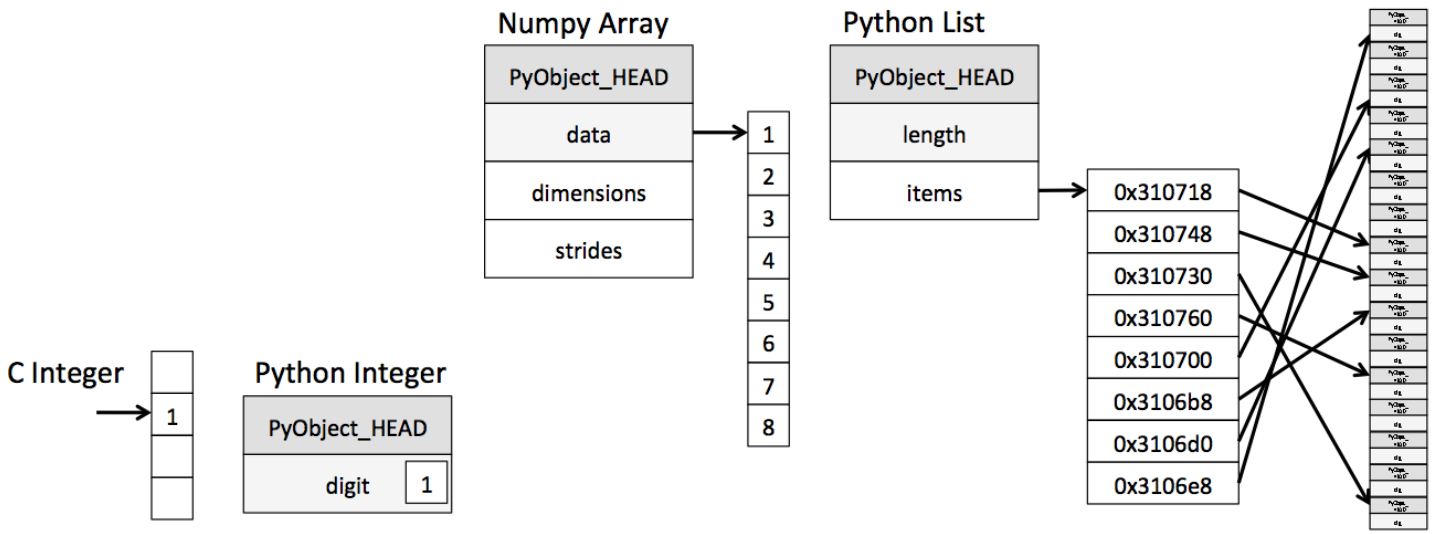
Pourquoi Optimiser

Pseudocode Python, c=a+b

```
#Python code
#a = 1
#b = 2
#c = a + b

Assign 1 to a
   1a. Set a->PyObject_HEAD->typecode to integer
   1b. Set a->val = 1

Assign 2 to b
   2a. Set b->PyObject_HEAD->typecode to integer
   2b. Set b->val = 2
call binary_add(a, b)
   3a. find typecode in a->PyObject_HEAD
   3b. a is an integer; value is a->val
   3c. find typecode in b->PyObject_HEAD
   3d. b is an integer; value is b->val
   3e. call binary_add<int, int>(a->val, b->val)
   3f. result of this is result, and is an integer.

Create a Python object c
   4a. set c->PyObject_HEAD->typecode to integer
   4b. set c->val to result
```

Pourquoi optimiser

Gestion des acces mémoire



Une solution : Numpy ?

Limitations Numpy

- Utilisation de tableaux intermédiaires
- Très lent dans la gestion des index (rapide dans les opérations vectorielles)
- Pas de parallélisation possible
- Pas de vectorisation possible
- utilisation des blasmkl pour l'algèbre linéaire, allocation de mémoires contigues !

Optimisation

Cython

- Du code python annoté et transformé en C.
- Performance proche du C.
- Parallélisme explicite.
- Peut appeler du code C.
- Perte de compatibilité avec Python ...
- Limitation des types/instructions gérés.

Optimisation

Cython

```
cdef inline int mandel(double real, double imag, int max_iterations=20):
    cdef double z_real = 0., z_imag = 0.
    cdef int i
    for i in range(0, max_iterations):
        z_real, z_imag = ( z_real*z_real - z_imag*z_imag + real,
                                           2*z_real*z_imag + imag )
        if (z_real*z_real + z_imag*z_imag) >= 4:
            return i
    return -1
```

Optimisation

PyPy

- Just–In–Time compiler
- objectif futur : un support complet du language.
- Support limité des modules(incl. Numpy)
- Un projet Numpypy.

Optimisation

Cython

```
def mandel(real, imag, max_iterations=20):
    z_real, z_imag = 0., 0.
    for i in range(0, max_iterations):
        z_real, z_imag = ( z_real*z_real - z_imag*z_imag + real,
                                           2*z_real*z_imag + imag )
        if (z_real*z_real + z_imag*z_imag) >= 4:
            return i
    return -1
```

Optimisation

Numba (// parakeet)

- Just-In-Time compiler.
- Support commercial.
- GPGPU backend.
- comportement excellent sur les boucles explicites.
- Basé sur LLVM.

Etude de cas simple

Mandelbrot avec Numba 1

```
import numpy as np
from pylab import imshow, show
from timeit import default_timer as timer
from numba import autojit

@autojit
def mandel(x, y, max_iters):
#Given the real and imaginary parts of a complex number, determine if it is a
      candidate for membership in the Mandelbrot set given a fixed number of
      iterations.
    c = complex(x, y)
    z = 0.0j
    for i in range(max_iters):
        z = z*z + c
        if (z.real*z.real + z.imag*z.imag) >= 4:
            return i
        return max_iters
```

Etude de cas simple

Mandelbrot avec Numba 2

```
@autojit
def create_fractal(min_x, max_x, min_y, max_y, image, iters):
    height = image.shape[0]
    width = image.shape[1]
    pixel_size_x = (max_x - min_x) / width
    pixel_size_y = (max_y - min_y) / height
    for x in range(width):
        real = min_x + x * pixel_size_x
        for y in range(height):
            imag = min_y + y * pixel_size_y
            color = mandel(real, imag, iters)
            image[y, x] = color
image = np.zeros((1024, 1536), dtype = np.uint8)
create_fractal(-2.0, 1.0, -1.0, 1.0, image, 20)
imshow(image)
show()
```

Optimisation

Pythran

```
# ## cmd pour "precompiler le code pythran !
# ## pythran -fopenmp  -march=corei7-avx pythan.py -o pythran.so
# ## utilisé ensuite comme un import classique python

#pythran export mandel(float, float, int)
def mandel(real, imag, max_iterations=20):
    z_real, z_imag = 0., 0.
    for i in range(0, max_iterations):
        z_real, z_imag = ( z_real*z_real - z_imag*z_imag + real,
                                                2*z_real*z_imag + imag )
        if (z_real*z_real + z_imag*z_imag) >= 4:
            return i
    return -1
```

Optimisation

Pythran

- Ahead-Of-Time compiler.
- Compile et transforme les modules python en binaires natifs.
- Support d'une grande partie de Python (exception, generator, named parameters, ...).
- Nécessite l'export de fonction annoté (commentaires).

Optimisation

Exemple de gain avec Numba

- python mandelbrot_numba.py :
  Mandelbrot created in 1.692088 s.

Optimisation

Autres possibilités ...

- Multithread/openMP
- f2py,ctypes
- Multiprocessing/MPI
- Acceleration GPGPU, pycuda/pyopencl

Questions ?