M1BI, M2BI, M2ISDD Septembre 2017

Quelques modules d'intérêt en Python

Patrick Fuchs

Université Paris Diderot

Contributeur : Pierre Poulain, Thibault Tubiana, Romain Retureau

PLAN

1. Introduction
2. Numpy
3. Biopython
4. Programmation orientée objet
5. Tkinter
6. Autres modules d'intérêt
7. A suivre demain…

PLAN

1. Introduction
2. Numpy
3. Biopython
4. Programmation orientée objet
5. Tkinter
6. Autres modules d'intérêt
7. A suivre demain…

Introduction

- Langage Python:
  - Langage interprété à bytecode
  - Langage orienté objet (classes)
  - Multi-plateforme (Linux, Windows, Mac…)
  - Nombreuses bibliothèques
    - Tkinter : fenêtres graphiques
    - Biopython : bioinformatique
    - Numpy : manipulation de vecteurs, matrices, algèbre linéaire…

    etc... Il existe sûrement une bibliothèque pour votre problème !
- But du cours: petit tour d'horizon de quelques bibliothèques d'intérêt en Bioinfo

Mais avant cela…

Listes de compréhension

```
>>> [x**2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> [2**i for i in range(13)]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]

>>> [x for x in range(30) if x % 3 == 0 and x > 12]
[15, 18, 21, 24, 27]

>>> noprimes = [j for i in range(2, 8) for j in range(i*2,
50, i)]
>>> [x for x in range(2, 50) if x not in noprimes]
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```


PLAN

1. Introduction
2. Numpy (Scipy)
3. Biopython
4. Programmation orientée objet
5. Tkinter
6. Autres modules d'intérêt
7. A suivre demain…

Numpy

- Numpy = Numerical Python
- Bibliothèque de base pour le calcul numérique:
  - Manipulation d'objets de type 'array' à N-dimension(s)
  - Fonctions basiques d'algèbre linéaire
  - Fonctions basiques de Transformée de Fourier
  - Générateur de nombres aléatoires sophistiqué
- URL de Numpy (*anciennement 'Numeric'*):
  site officiel : http://www.numpy.org/
- Pour le Graphisme: utiliser d'autres bibliothèques (matplotlib, rpy, gnuplot.py)

Scipy (Scientific Tools for Python)

- Bibliothèque (basée sur numpy) permettant d'effectuer de nombreux calculs ou opérations scientifiques:
  - Statistiques
  - Optimisation
  - Intégration numérique
  - Algèbre linéaire
  - Transformée de Fourier
  - Traitement du signal
  - Traitement d'image
  - Algorithmes génétiques
  - etc…
- URL de Scipy: https://www.scipy.org/

Objet array

- Importation du module numpy :

```
>>> import numpy as np
…
```

- Définition d'un vecteur :

```
>>> v = np.array((1,2,3,4,5))
>>> v
array([1, 2, 3, 4, 5])
```

- Définition automatique d'un vecteur d'entiers :

```
>>> v = np.arange(10)# équivalent de range
>>> v
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```


Objet array (2)

- Opération vectorielle

```
>>> v = np.arange(10)
>>> v + 0.1
array([ 0.1,  1.1,  2.1,  3.1,  4.1,  5.1,  6.1,  7.1,
   8.1,  9.1])
>>>
```

- Redimensionner un objet array

```
>>> v = np.arange(4)
>>> np.reshape(v,(2,2)) # l'array v doit avoir son nombre
array([[0, 1],          # d'éléments compatible avec la
       [2, 3]])         # nouvelle taille (matrice 2*2)
>>> v = np.arange(9)
>>> np.resize(v,(3,4))  # l'array v peut contenir un
array([[0, 1, 2, 3],    # nombre quelconque d'éléments
       [4, 5, 6, 7],
       [8, 0, 1, 2]])
```


Objet array (3)

- Création d'une matrice de 0

```
>>> np.zeros((3,3))
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

- Création d'une matrice de 1

```
>>> np.ones((2,2))
array([[ 1.,  1.],
       [ 1.,  1.]])

>>> np.ones((2,2),int)
array([[1, 1],
       [1, 1]])
```

Objet array (4)

- Création d'une matrice diagonale (matrice identité)

```
>>> np.eye(4)
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

Objet array (5)

- Récupération de la taille d'un objet array

```
>>> matrice = np.reshape(np.arange(81),(9,9))
>>> np.shape(matrice)
(9, 9)
>>> print(matrice.shape)
(9, 9)
```

- Récupération de ligne/colonne/élément d'une matrice

```
>>> a = np.resize(np.arange(9),[3,3])
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> a[1,:] # récupération de la ligne 1 (l'index commence à 0)
array([3, 4, 5])
>>> a[:,1] # récupération de la colonne 1 (idem)
array([1, 4, 7])
>>> a[2,2] # élément à la ligne 2/colonne 2 (idem)
8
```


Algèbre linéaire

- Définition d'une matrice 2*2

```
>>> a = np.array([[1,2],[3,4]])
>>> a
array([[1, 2],
       [3, 4]])
```

- Multiplication de Matrice

```
>>> np.dot(a,a)
array([[ 7, 10],
       [15, 22]])

>>> a.dot(a)
array([[ 7, 10],
       [15, 22]])
```

Attention !

multiplication de matrice ≠ produit par élément

```
>>> a*a
array([[ 1,  4],
       [ 9, 16]])
```


Algèbre linéaire (2)

- Inversion de Matrice

```
>>> import numpy.linalg as ln # importation du module
                              # d'algèbre linéaire
>>> a = np.array([[1,2],[3,4]])
>>>
>>> ln.inv(a)
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

Rappel : $\mathbf{M} \cdot \mathbf{M}^{-1} = \mathbf{I}$
(*le produit matriciel de M par son inverse est égale à la matrice idéntité*)

Algèbre linéaire (2)

- Importation du module d'algèbre linéaire

```
>>> import numpy.linalg as ln
```

- Définition d'une matrice 3*3

```
>>> a = np.resize(np.arange(9),(3,3))
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Rappel : $\mathbf{M} \cdot \mathbf{v}_i = \lambda_i \cdot \mathbf{v}_i$ (λ_i et $\mathbf{v}_i$ sont respectivement les valeurs et vecteurs propres de la matrice M)

- Valeurs propres

```
>>> eigenvalues(a)
array([  1.33484692e+01,  -1.34846923e+00,  -1.14063163e-15])
```

- Vecteurs (et valeurs) propres

```
>>> eigenvectors(a)
(array([  1.33484692e+01,  -1.34846923e+00,  -1.14063163e-15]),
 array([[-0.16476382, -0.50577448, -0.84678513],
        [-0.79969966, -0.10420579,  0.59128809],
        [ 0.40824829, -0.81649658,  0.40824829]]))
```


Objets array-2D ≠ matrice

```
>>> a = np.resize(np.arange(9),(3,3))
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> np.matrix(a)
matrix([[0, 1, 2],
        [3, 4, 5],
        [6, 7, 8]])
>>>
```

Copies d'arrays

```
>>> a = np.resize(np.arange(9)\
                  ,(3,3))
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> b = np.array(a)
>>> b[2,2] = -155
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> b
array([[   0,    1,    2],
       [   3,    4,    5],
       [   6,    7, -155]])
```

```
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> b = a
>>> b[-2,2] = -155
>>> a
array([[   0,    1,    2],
       [   3,    4, -155],
       [   6,    7,    8]])
>>> b
array([[   0,    1,    2],
       [   3,    4, -155],
       [   6,    7,    8]])
```

Masques d'indices

```
>>> a = np.resize(np.arange(9),(3,3))
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> a > 6
array([[False, False, False],
       [False, False, False],
       [False,  True,  True]], dtype=bool)
>>> a [a > 6]
array([7, 8])
>>> a[a > 6] = -157
>>> a
array([[   0,    1,    2],
       [   3,    4,    5],
       [   6, -157, -157]])
>>>
```

etc

```
>>> dir(np)
```

…

PLAN

1. Introduction
2. Numpy
3. Biopython
4. Programmation orientée objet
5. Tkinter
6. Autres modules d'intérêt
7. A suivre demain…

Biopython

- Fonctionnalités de biopython :
  - Travail avec les Séquences
  - Parser de fichiers de banques biologiques
  - BLAST automatiques
  - Clustalw
  - Parser de PDB

  etc…

Site: http://biopython.org/

Tutorial:
http://biopython.org/DIST/docs/tutorial/Tutorial.html

Travail avec les séquences

- But: gérer la séquence elle-même plus d'autres informations d'intérêt pour les biologistes
- Classe Seq:
  - 2 attributs importants:
    - data: séquence elle-même
    - alphabet: type de séquence (ADN, ARN, Prot *etc…*)
  - Avantage: on peut traiter par exemple 2 types d'ADN (non ambigu [ATGC], ou ambigu ATGCNRW *etc…*)

Travail avec les séquences (2)

- Définition d'un alphabet
  (Bio.Alphabet IUPAC contient des définitions d'alphabets classiques pour les ADN, ARN, Prot…)

```
>>> from Bio.Alphabet import IUPAC
```

- Définition d'un objet Séquence

```
>>> my_alphabet = IUPAC.unambiguous_dna
>>> from Bio.Seq import Seq
>>> my_seq =
   Seq('CATCCCTTCGATCGGGGCTATAGCTAGC',my_alphabet)
>>> my_seq
Seq('CATCCCTTCGATCGGGGCTATAGCTAGC',
   IUPACUnambiguousDNA())
>>> print(my_seq)
CATCCCTTCGATCGGGGCTATAGCTAGC
```


Travail avec les séquences (3)

- Comportement analogue aux chaînes de caractères

```
>>> print(my_seq[4:12])
CCTTCGAT
>>>
>>> print(len(my_seq))
28
>>> new_seq = my_seq[0:5]
>>> new_seq
Seq('CATCC', IUPACUnambiguousDNA())
>>> my_seq + new_seq
Seq('CATCCCTTCGATCGGGGCTATAGCTAGCCATCC',
   IUPACUnambiguousDNA())
>>>
```


Transcription et Traduction

- Transcription en ARN:

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG",
   IUPAC.unambiguous_dna)
>>> messenger_rna = coding_dna.transcribe()
>>> messenger_rna
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPACUnambiguousRNA())
```

- Traduction en Protéine:

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> messenger_rna = Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG",
   IUPAC.unambiguous_rna)
>>> messenger_rna
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPACUnambiguousRNA())
>>> messenger_rna.translate()
Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
```

(traduit aveuglément même avec des codons stop)

Rechercher et Parser des fichiers de banques biologiques

- Fonctionne sur les banques de données suivantes:
  - ExPASy
  - Entrez (NCBI)
  - PubMed (NCBI)
  - SCOP

  etc

Requête sur Entrez@NCBI

```
$ export http_proxy="http://ewebproxy.univ-paris-diderot.fr:3128"
$ export https_proxy="http://ewebproxy.univ-paris-diderot.fr:3128"
$python3
>>> from Bio import Entrez
>>> from Bio import SeqIO
>>> Entrez.email = "A.N.Other@example.com"
>>> handle = Entrez.efetch(db="nucleotide",
   rettype="fasta", retmode="text", id="6273291")
>>> seq_record = handle.read()
>>> handle.close()
>>> seq_record
'>AF191665.1 Opuntia marenae rpl16 gene; chloroplast gene
   for chloroplast product, partial intron
   sequence\nTATACATTAAAGGAGGGGGATGC[…]TGTC\nCGGTTCTGTAGT
   AGAGATGGAATTAAGAAAAAACCATCAACTATAACCCCAAAAGAACCAGA\n\n
   '
```


BLAST automatique

Requete BLAST

```
>>> from Bio.Blast import NCBIWWW
>>> result_handle = NCBIWWW.qblast("blastn", "nt", "8332116")
```

Pb : ne fonctionne pas toujours…

Sauver les résultats dans un fichier XML

```
>>> save_file = open("my_blast.xml", "w")
>>> save_file.write(result_handle.read())
>>> save_file.close()
>>> result_handle.close()
```

Parser les résultats

```
>>> result_handle = open("my_blast.xml")
>>> from Bio.Blast import NCBIXML
>>> blast_record = NCBIXML.read(result_handle)
>>> E_VALUE_THRESH = 0.04
```

Pb : le NCBI change souvent le format de sortie de BLAST…

à suivre...

Parsing de la requête BLAST

```
for alignment in blast_record.alignments:
   for hsp in alignment.hsps:
     if hsp.expect < E_VALUE_THRESH:
       print '****Alignment****'
       print 'sequence:', alignment.title
       print 'length:', alignment.length
       print 'e value:', hsp.expect
       print hsp.query[0:75] + '...'
       print hsp.match[0:75] + '...'
       print hsp.sbjct[0:75] + '...'
```

à suivre...

Résultat du parsing BLAST

```
****Alignment****
sequence: gi|224094601|ref|XM_002310151.1| Populus trichocarpa predicted
protein, mRNA
length: 1089
e value: 6.83144e-88
AAAATGGGGAGAGAAATGAAGTACTTGGCCATGAAAACTGATCAATTGGCCGTGGCTAATATGATCGATTCCGAT...
||||||||||    ||| |||  |||   ||||| |||||| |    | |   |||||| | || || |||||...
AAAATGGGGAGG---ATGGAGTTTTTGAAGATGAAGACTGATGATGAAGTCAGCGCTAATTTAATTGAGTCCGAT...

****Alignment****
sequence: gi|359495761|ref|XM_002274845.2| PREDICTED: Vitis vinifera
uncharacterized LOC100267774 (LOC100267774), mRNA
length: 853
e value: 1.50473e-83
TGAACAGAAAATGGGGAGAGAAATGAAGTACTTGGCCATGAAAACTGATCAATTGGCCGTGGCTAATATGATCGA...
||||  ||||||||||||    ||| ||||  |||| ||||||||||||    |  |    | ||| ||||  |...
TGAAACGAAAATGGGGAGG---ATGGAGTATCTGGCTATGAAAACTGATCCC--GAACCAACCCAAT-TGATTAA...

etc
```

Requête sur la Genbank

```
>>> from Bio import Entrez
>>> Entrez.email = "A.N.Other@example.com"

>>> handle = Entrez.efetch(db="nucleotide", id="186972394",
rettype="gb", retmode="text")
>>> print(handle.read())

LOCUS       EU490707               1302 bp    DNA     linear   PLN 05-MAY-2008
DEFINITION  Selenipedium aequinoctiale maturase K (matK) gene, partial cds;
            chloroplast.
ACCESSION   EU490707
VERSION     EU490707.1  GI:186972394
KEYWORDS    .
SOURCE      chloroplast Selenipedium aequinoctiale
[...]
ORIGIN
        1 atttttacg aacctgtgga aatttttggt tatgacaata aatctagttt agtacttgtg
[...]
     1261 ttaggttcgg gattattaga agaattcttt atggaagaag aa
//
```


PLAN

1. Introduction
2. Numerical Python
3. Biopython
4. Programmation orientée objet
5. Tkinter
6. Autres modules d'intérêt
7. A suivre demain…

Très brève introduction à la programmation orientée objet

- Une classe définit des objets qui sont des instances (~ représentants) de cette classe
- Les objets possèdent des attributs (~variables) et des méthodes (~ fonction) associés à cette classe
- Propriétés de la programmation orientée objet :
  - Encapsulation (interface 'publique', implémentation de l'interface 'privée')
  - Polymorphisme : capacité à transformer les opérateurs standards (e.g. +, * *etc…*) en fonction du contexte
  - Héritage multiple : possibilité de créer des sous-classes héritant des spécialisations des classes mères

Les Classes Python

Les classes Python définissent des objets:

– Les objets sont des instances des classes

```
class Atom:
  def __init__(self,atno,x,y,z):
    self.atno = atno
    self.position = (x, y, z)
```

__init__ est le constructeur par défaut

self désigne l'objet lui même

Exemple: classe atome

```
class Atome:
    # constructeur
    def __init__(self,atno,x,y,z):
        self.atno = atno
        self.position = (x,y,z)
    # definit le comportement avec print
    def __repr__(self):
        return '{:4d} {:10.4f} {:10.4f}
  {:10.4f}'.format(self.atno,self.position[0],self.p
  osition[1],self.position[2])

>>> at = Atome(6,0.0,1.0,2.0)
>>> print(at)
6      0.0000      1.0000      2.0000
>>>
```


Classe atome

- Ecrase le constructeur par défaut
- Définit des variables de classes (atno, position) qui sont persistantes et locales à l'objet
- Meilleur moyen de gérer la mémoire
  - passage de l'objet, donc de toutes ses données encapsulées, au lieu d'une longue liste d'arguments
  - programmes plus 'propres'
- Ecrase le comportement avec print

Classe molecule

```
class Molecule:
    #constructeur
    def __init__(self,name='Generic'):
        self.name = name
        self.atomlist = []
    # nouvelle methode
    def addatom(self,atome):
        self.atomlist.append(atome)
    #  comportement avec print
    def __repr__(self):
        chaine = 'This mlc is named {}\n'.format(self.name)
        chaine += 'It has {} atoms\n'.format(len(self.atomlist))
        for atom in self.atomlist:
            chaine += str(atom) + '\n'
        return chaine
```


Classe molecule (2)

```
>>> mol = Molecule('water')
>>> at = atome(1,0.,0.,0.)
>>> at
1      0.0000      0.0000      0.0000
>>> mol.addatom(at)
>>> mol.addatom(atome(2,0.,0.,1.))
>>> mol.addatom(atome(3,0.,1.,0.))
>>> print mol
This mlc is named water
It has 3 atoms
1      0.0000      0.0000      0.0000
2      0.0000      0.0000      1.0000
3      0.0000      1.0000      0.0000

>>>
```

Pour aller plus loin…

- Autres attributs :
  __add__(self,other) : opérateur '+'
  __mul__(self,number) : opérateur '*'
  __del__(self) : destructeur par défaut
  __len__(self) : opérateur len
  __getslice__(self,low,high) : récupérer des tranches
  etc...
- Livre: Gérard Swinnen
  http://inforef.be/swi/python.htm

PLAN

1. Introduction
2. Numpy
3. Biopython
4. Programmation orientée objet
5. Tkinter
6. Autres modules d'intérêt
7. A suivre demain…

Le module Tkinter

- Tk (Tool Kit) : jeu d'objets graphiques ou encore widgets (window gadget)
- Conçu au départ pour être piloté pat Tcl (Toolkit Control Language)
  - Tcl limité ?
  - possible de piloter Tk depuis Perl ou Python
- Tkinter est l'interface Python / Tk
  - facile à utiliser (selon certains auteurs…)
  - puissant
- Permet de développer des programmes avec des interfaces graphiques (GUI)
- Par défaut dans les distributions python mais nécessite d'avoir installé Tk et Tcl (http://www.scriptics.com)

Un premier exemple simple

```
from Tkinter import *

racine = Tk()

texte = Label(racine,text="Hi there !",fg="red")
texte.pack()

bouton = Button(racine, text="Quit",
      command=racine.destroy)
bouton.pack()

racine.mainloop()
```

Premier exemple commenté

- importation du module Tkinter

```
from Tkinter import *
```

- création d'une instance d'un objet graphique Tk (classe Tk = Toplevel widget of Tk)
  racine est un objet représentant la fenêtre de base

```
racine = Tk()
```

- Création d'un objet (widget) de la classe Label(); il sera contenu dans le widget maître 'racine'

```
texte = Label(racine,text="Hi there !",
  fg="red")
```

- Appel de la méthode pack (ajuste la taille de la fenêtre maître)

```
texte.pack()
```


Premier exemple commenté (2)

- Création d'un second widget 'esclave' dans le widget 'racine', de la classe 'Button' (l'attribut command indique l'action à effectuer en cas de click)

```
bouton = Button(racine, text="Quit",
  command=racine.destroy)
```

- Appel de la méthode pack

```
bouton.pack()
```

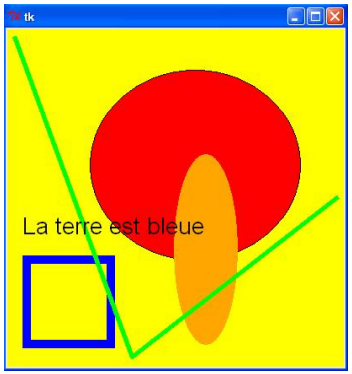
- Démarrage du gestionaire d'évènement (obligatoire en script, inutile en interactif)

```
racine.mainloop()
```


Widget Canvas (code)

```
from tkinter import *

##################
# Prog Principal #
##################
fenetre = Tk()
canevas = Canvas(fenetre, width=400, height=400, background='yellow')

canevas.create_oval(100,50,350,275, fill='red')
canevas.create_oval(200,150,275,375, fill='orange',width=0)
canevas.create_rectangle(25,275,125,375,outline='blue',width=10.0)

canevas.create_line(10,10, 150,390,395,200, width=5.0,fill="green")
canevas.create_text(20,235, font=("Geneva",21), text="La terre est
   bleue",anchor="w")

canevas.pack(expand=YES,fill=BOTH)
fenetre.mainloop()
```


Widget Canvas (exemple)



Le widget Canvas permet de construire un container pouvant accueillir des dessins

Widget Canvas (code commenté)

- Importation de Canvas facultative (déjà inclus dans Tkinter)

```
from Tkinter import *
```

- Instancie un objet Tk

```
fenetre = Tk()
```

- Définition du canevas - largeur, hauteur (en pixels) et couleur du fond - qui définit aussi la fenêtre

```
canevas = Canvas(fenetre, width=400, height=400,
  background='yellow')
```

- Ajoute une ellipse rouge avec une bordure noire

```
canevas.create_oval(100,50,350,275, fill='red')
```

- Ajoute une ellipse orange sans bordure (width = largeurDeLaBordure)

```
canevas.create_oval(200,150,275,375,
  fill='orange',width=0)
```


Widget Canvas (code commenté, suite)

- Ajoute un carré avec seulement une bordure bleue

```
canevas.create_rectangle(25,275,125,375,outline='blu
  e',width=10.0)
```

- Et une ligne verte, de largeur 5 pixels (x0,y0,x1,y1,x2,y2)

```
canevas.create_line(10,10, 150,390,395,200,
  width=5.0,fill="green")
```

- Finalement, on ajoute un peu de texte (texte en 20,235 ("west")

```
canevas.create_text(20,235,   # Position
          font=("Geneva",21), # Type de font
          text="La terre est bleue",anchor="w")
```

- On place le tout dans le canevas

```
canevas.pack(expand=YES,fill=BOTH)
```

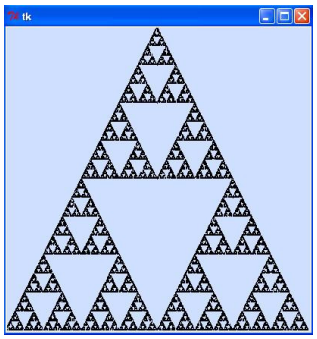
- Garde la figure à l'écran

```
fenetre.mainloop()
```


Classes de widgets de Tkinter

- Widgets
  - Button — boutton
  - Canvas — dessin (lignes, cercles, images *etc…*)
  - Checkbutton — Case à cocher
  - Entry — Champ d'entrée
  - Label — Texte
  - Listbox — Liste de choix
  - Menu — Menu déroulant ou 'pop up'
  - Menubutton — Pour implémenter les menus déroulants
  - Message — Texte
  - Radiobutton — Boutons radios
  - Scale — Curseur / règle
  - Text — Texte formaté
  - Toplevel — Fenêtre affichée séparément 'par-dessus'
- Méthodes pour positionner ces widgets
  - pack()
  - grid()
  - place()

Triangle de Sierpinski



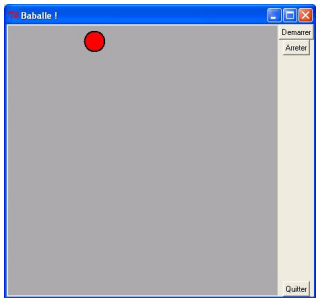
Exercices:

- comment créer le triangle de Sierpinski ?
  - définir les 3 sommets
  - partir du centre (création d'un petit cercle d'une pixelle de large)
  - choisir un sommet au hasard
  - se déplacer à mi-distance entre le point actuel et ce sommet, et y tracer un point
  - réitérer les 2 dernières actions 25000 à 30000 fois
- reproduire la même chose avec 4 sommets

Graphiques animés (une petite introduction)

- Pratique pour visualiser l'évolution de simulations
- Limité à la 2D (pour la 3D, plutôt se tourner vers des bibliothèques plus spécialisées : PyOpenGL)
- La méthode coords permet de bouger un objet créé dans un Widget Canvas (e.g. depuis un create_oval)
- Animation automatique: fonctions récursives
- Pour l'instant utilisation de variables globales…
- …MAIS avec les classes, possibilité de coder 'plus propre'

Une Baballe qui bouge !



- But:
  - faire bouger une balle dans un espace lorsque l'on clique sur 'Démarrer'
  - Arrêter la balle lorsque l'on clique sur arrêter
  - Quitter avec le boutton correspondant

Le code du Prog Principal

```
####### Programme principal #########
#var globales
x1, y1 = 10, 10 #coord ini
dx, dy = 15, 0  #pas du deplacement
flag = 0

#creation du widget principal (maitre)
fen1 = Tk()
fen1.title("Baballe !")

#creation des widgets esclaves (canevas + baballe + boutons)
can1 = Canvas(fen1,bg='dark grey',height=400,width=400)
can1.pack(side=LEFT)
oval1 = can1.create_oval(x1,y1,x1+30,y1+30,width=2,fill="red")
Button(fen1,text="Quitter",command=fen1.quit).pack(side=BOTTOM)
Button(fen1,text="Demarrer",command=start_it).pack()
Button(fen1,text="Arreter",command=stop_it).pack()

#demarrage du receptionnaire d'evenements (boucle principale)
fen1.mainloop()
```


L'idée : la fonction "récursive"

```
# fonction de deplacement
def move():
    "deplacement de la baballe !"
    global x1,y1,dx,dy,flag
    x1,y1 = x1 + dx, y1 + dy
    if x1 > 360:
        x1, dx, dy = 360, 0, 15
    if y1 > 360:
        y1, dx, dy = 360, -15, 0
    if x1 < 10:
        x1, dx, dy = 10, 0, -15
    if y1 < 10:
        y1, dx, dy = 10, 15, 0
    # modif de coordonnees
    can1.coords(oval1,x1,y1,x1+30,y
    1+30)
    # appel recursif
    if flag > 0:
        fen1.after(50, move)
        #boucler apres 50 ms
```

```
# fonction pour stopper
def stop_it():
    "arret de l'animation"
    global flag
    flag = 0

# fonction pour demarrer
def start_it():
    "demarrage de l'animation"
    global flag
    flag += 1 #preferable à
              # flag = 1
    if flag == 1:
        move()
```

PLAN

1. Introduction
2. Numpy
3. Biopython
4. Programmation orientée objet
5. Tkinter
6. Graphisme sous python et autres modules d'intérêt
7. A suivre demain…

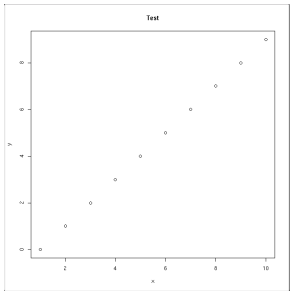
RPy (R from Python)

- Permet d'importer des fonctions R

```
>>> import rpy2.robjects as robjects
>>> l = list(range(10))
>>> robjects.r['plot'](l,l,main="Test",xlab="x",ylab="y")
```

liste python

appel d'une commande R



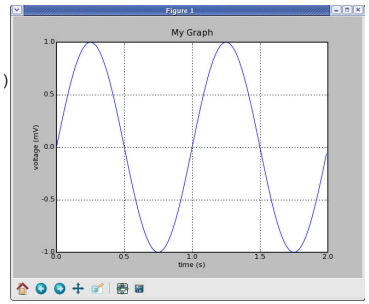
- URL: http://rpy.sourceforge.net/
- URL: http://cran.cict.fr/

(non installé en salle info)

Matplotlib

- Bibliothèque permettant de générer des graphiques (interactifs !)
  Exemple:

```
from pylab import *
t = arange(0.0, 2.0, 0.01)
s = sin(2*pi*t)
plot(t, s, linewidth=1.0)
xlabel('time (s)')
ylabel('voltage (mV)')
title('My graph')
grid(True)
show()
```



- URL : http://matplotlib.sourceforge.net/

Module subprocess

- gestion des entrées / sorties d'une commande Unix

https://docs.python.org/3/library/subprocess.html

Module subprocess (2)

```
import subprocess
command = "wc -l"
data = """>sp|Q41560|HS16B_WHEAT 16.9 kDa class I heat shock protein
MSIVRRTNVFDPFADLWADPFDTFRSIVPAISGGGSETAAFANARMDWKETPEAHVFKAD
LPGVKKEEVKVEVEDGNVLVVSGERTKEKEDKNDKWHRVERSSGKFVRRFRLLEDAKVEE
VKAGLENGVLTVTVPKAEVKKPEVKAIQISG
"""
proc = subprocess.Popen(command, shell = True,
stdout = subprocess.PIPE, stderr = subprocess.PIPE,
stdin = subprocess.PIPE)
# contenu de la sortie standard
# et de la sortie d'erreur standard
(out, err) = proc.communicate(data)
print("===Entree standard :")
print(data)
print("===Sortie standard :")
print(out)
print("===Sortie d'erreur standard :")
print(err)
# affiche le code de sortie (0 = OK)
print("===Code de sortie :")
print(proc.wait())
```

A vérifier !

Sortie de subprocess

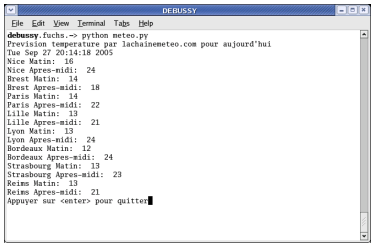
```
===Entree standard :
>sp|Q41560|HS16B_WHEAT 16.9 kDa class I heat shock protein
MSIVRRTNVFDPFADLWADPFDTFRSIVPAISGGGSETAAFANARMDWKETPEAHVFKAD
LPGVKKEEVKVEVEDGNVLVVSGERTKEKEDKNDKWHRVERSSGKFVRRFRLLEDAKVEE
VKAGLENGVLTVTVPKAEVKKPEVKAIQISG
===Sortie standard :
4
===Sortie d'erreur standard :
===Code de sortie :
0
```


Autres pistes

- Le module **urllib2** (permet de charger dynamiquement des pages HTML)

  e.g. meteo.py

  interroge la météo en temps réel !



- **gnuplot.py** permet d'importer gnuplot dans python
- **pymol** possède un interpréteur python embarqué…
- idem pour **vmd**

etc...

PLAN

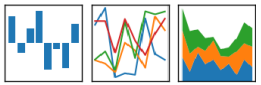
1. Introduction
2. Numpy
3. Biopython
4. Programmation orientée objet
5. Tkinter
6. Autres modules d'intérêt
7. A suivre demain…

A suivre demain…

Panda

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Ipython notebook

IP[y]: IPython
Interactive Computing

Bonnes pratiques : PEP8, pylint

FIN

à vous !