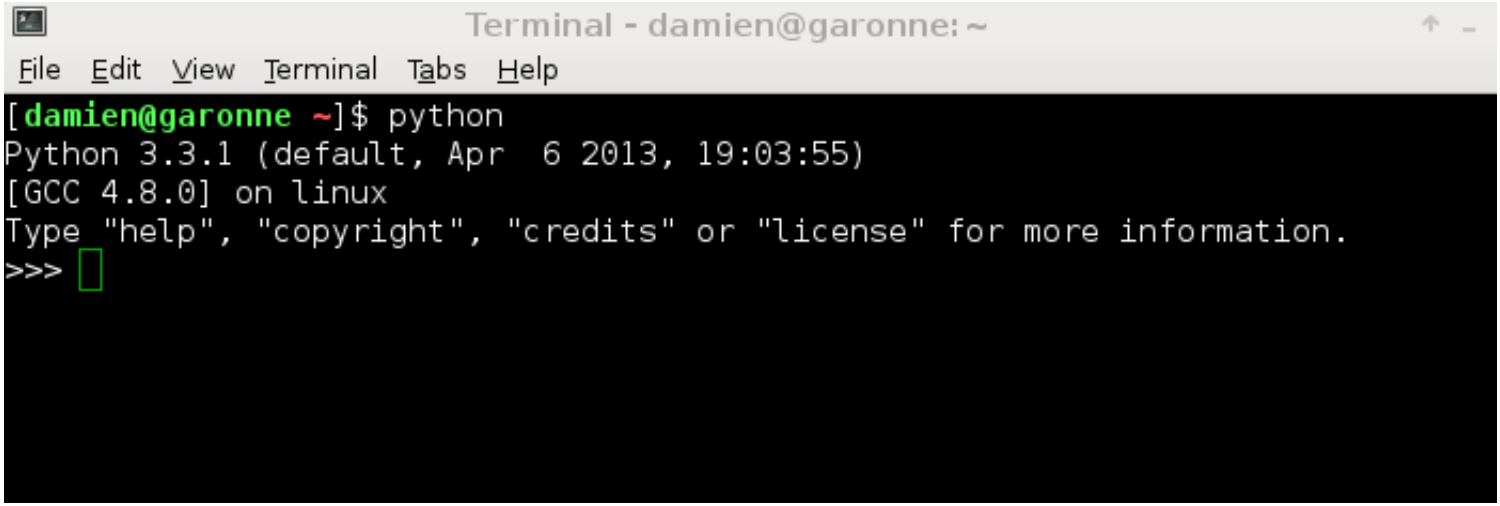
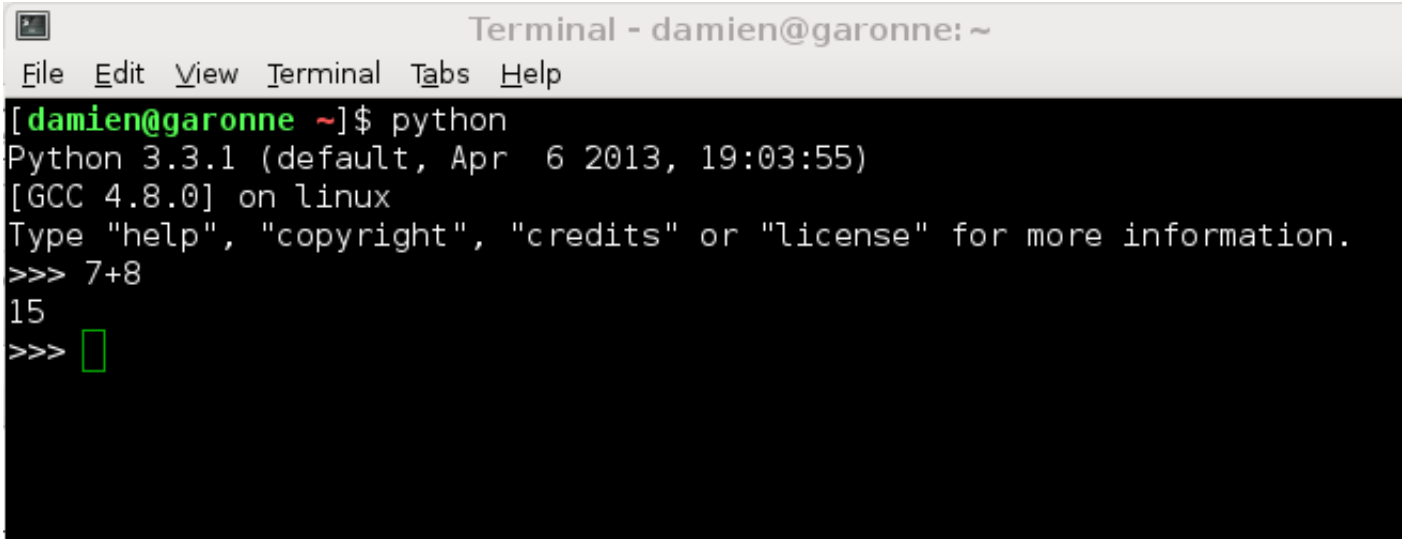
Introduction à la programmation

Python

Premier "programme"



Premier "programme"



Commandes

Notion de variables:

```
a=7
b=2
a+8+b*b

> 17
```

a est une **variable** (qui vaut 7)
b est une **variable** (qui vaut 2)

Commandes

Notion de variables:

```
a=4
b=a+1
b=b+2

print(b)
```

```
> 7
```

Commandes

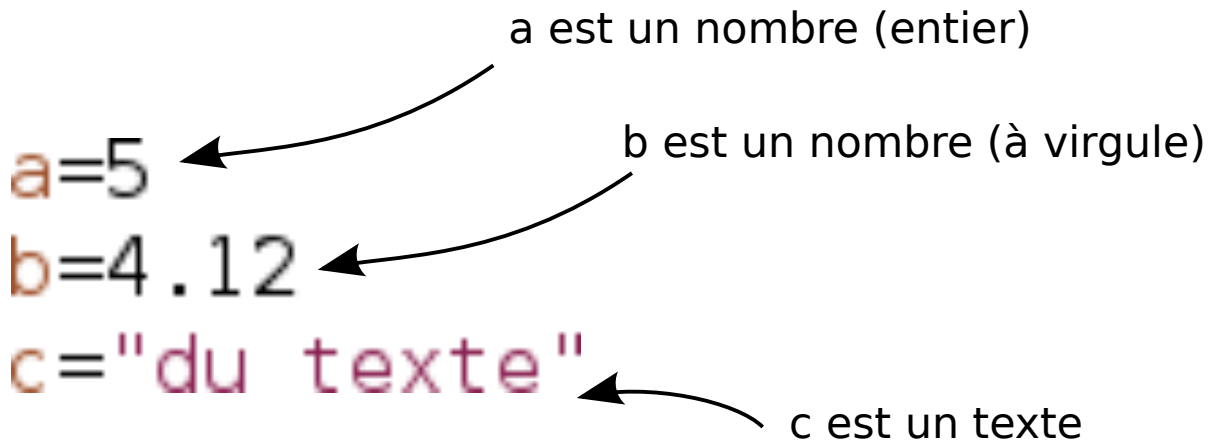
Affichage à l'écran:

```
print("le resultat de 2+2 vaut",2+2)
```

> le resultat de 2+2 vaut 4

Commandes

Types de variables:



a+b OK

a+c

unsupported operand type(s) for +: 'int' and 'str'

Commandes

Types de variables:

```
a=7
b=2
c=-b/a;
print("la solution de l'equation ",a,"x + ",b,"=0 vaut",c)
```

Commandes

Variable nombre/texte:

```
mon_texte_1="4+7"
mon_texte_2="2+2"
valeur_1=4+7
valeur_2=2+2

mon_texte_3=mon_texte_1+mon_texte_2
valeur_3=valeur_1+valeur_2

print(mon_texte_3)
print(valeur_3)
```

ceci est du texte

ceci est un nombre

```
> 4+72+2
> 15
```

Commandes

Variable nombre/texte:

transforme un nombre en texte
(str=string)

```
variable_nombre=4+8
variable_texte=str(variable_nombre);

variable_texte=variable_texte+"78"

print(variable_texte)
```

> 1278

L'aide

Pour obtenir de l'aide sur une fonction:

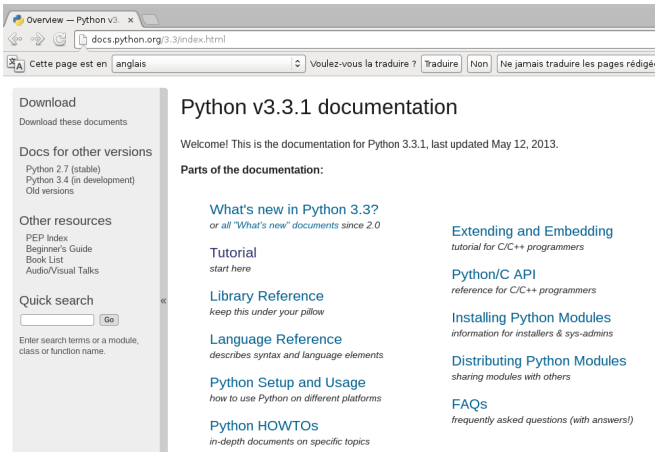
help(*nom_fonction***)**

ex. help(pow)

Site web:

http://docs.python.org/2/index.html

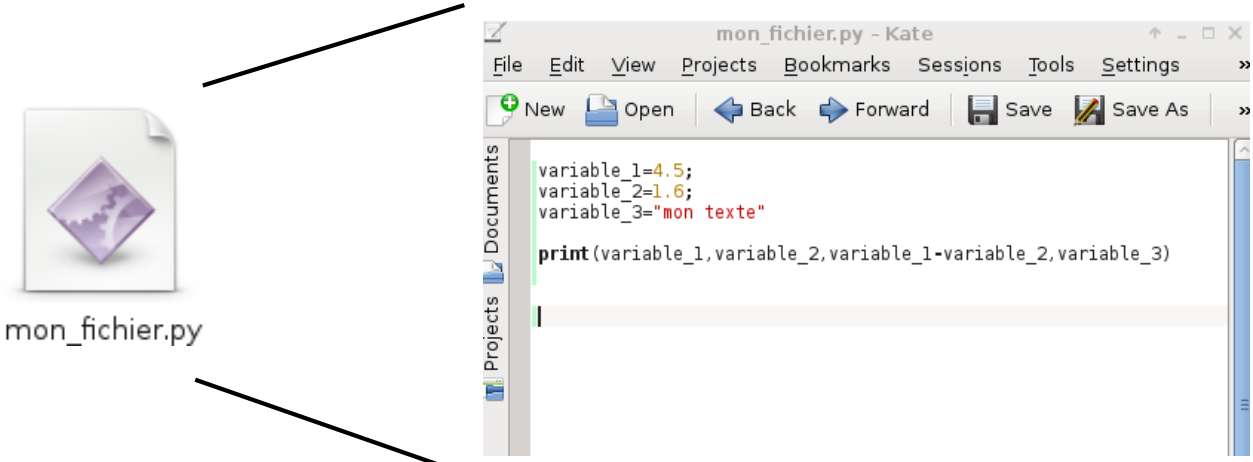
http://docs.python.org/3/index.html



Ecriture dans un fichier

Ecrire ligne à ligne est fastidieux ...

On écrit d'abord dans un fichier texte



(.py = fichier texte lisible par Python)

On lance Python sur le fichier

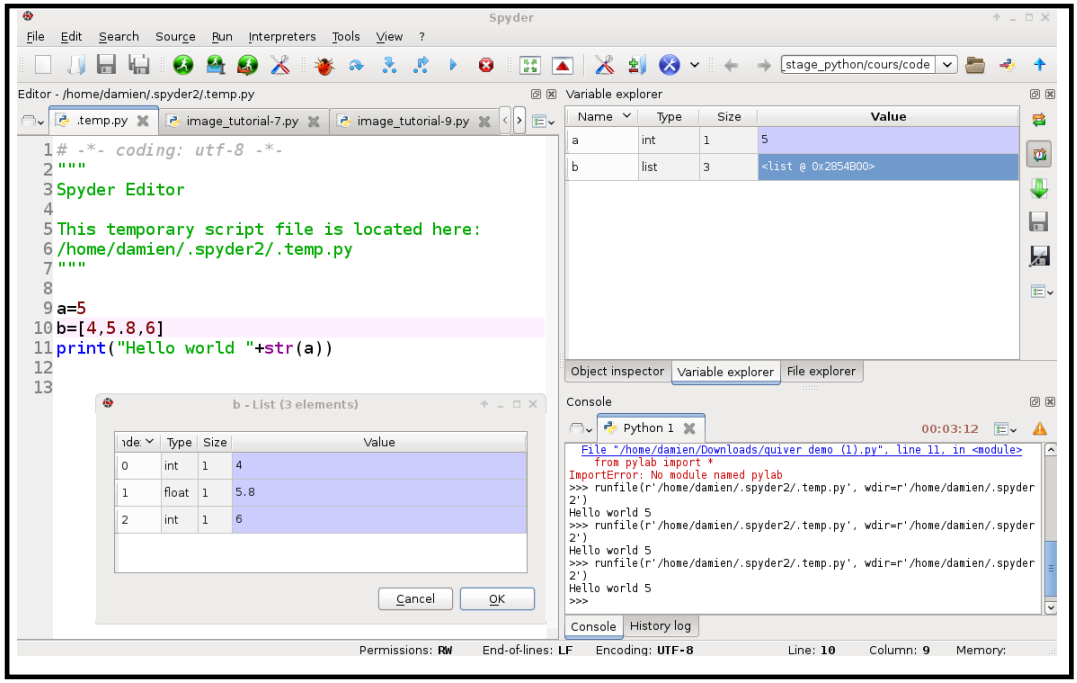
```
[damien@damien_pc ~/work/2012_2013_teaching
ython/cours/code]$ python mon_fichier.py
4.5 1.6 2.9 mon texte
```

Editeur Python

Editeur de texte (attention à l'indentation)

- Linux: Kate
- Window: par defaut, pyscripter

Editeur type Matlab: **Spyder**



Python: le langage

Création en 1990 (C ~ 1973)

Scripts, manipulation texte, pas de scientifique

Module Numpy en 2005

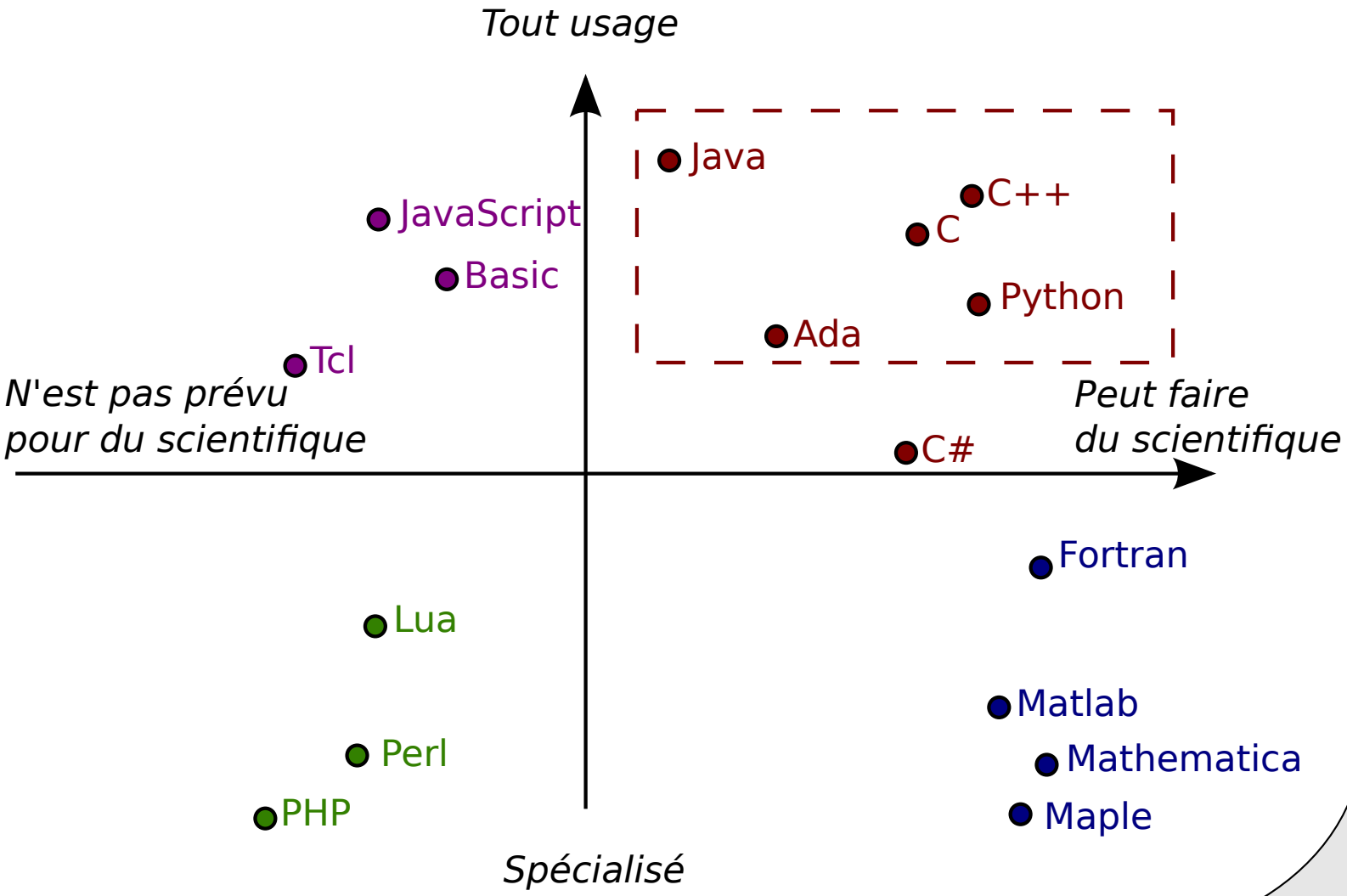
Developpement du calcul scientifique

Python 2.0 en 2000
Python 3.0 en 2009

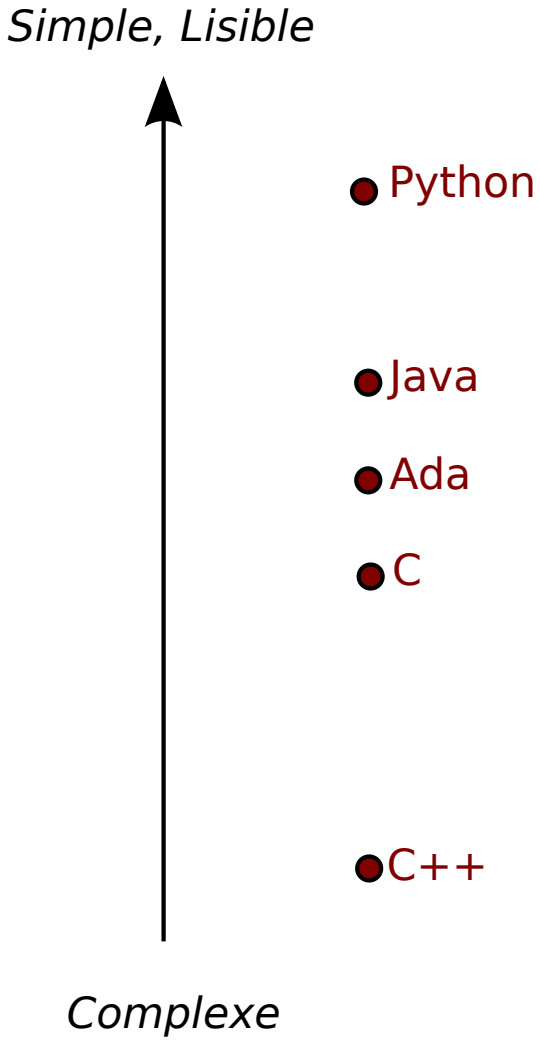
Python devient un acteur majeur du monde du calcul scientifique

- beaucoup de modules (scientifique, visualisation, etc)
- lisible
- simple à écrire
- language haut niveau
- potentiellement optimisable

Python: positionnement

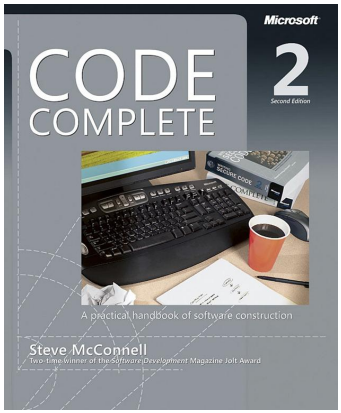


Python: positionnement



Python: Les + / -

+ Langage très lisible



Lisibilité d'un code = Le + Important

Un code est beaucoup plus lue qu'écrit
Le code est sa propre documentation
Erreur facilement détectable = gain de temps

+ Langage très lisible

Java

```java
public class MonTest{
    public static void main(string[] args){
        for(int i=0;i<10000;i++)
            System.out.println(i);
    }
}
```

Python

```python
for x in range(10000):
    print(i)
```

Ce qui est simple
s'écrit simplement

+ Langage très lisible

C++ : occurence dans un conteneur

La généricité
s'écrit "naturellement"

```cpp
template <typename T>
int count_x(const T& conteneur,const typename T::value_type& y)
{
  int compteur=0;
  for(typename T::const_iterator it=conteneur.begin();it!=conteneur.end();++it)
    if(*it==y)
      compteur++;
  return compteur;
}

int main()
{
  std::string a[]={"maison","toto","tata","histoire","maison","royaume","maison"};

  std::list<std::string> liste;
  std::copy(&a[0],&a[7],std::back_inserter(liste));

  std::cout<<count_x(liste,"maison")<<std::endl;
}
```

Commentaires indispensables!!

Python

```python
def count_x(vecteur,x):
  counter=0;
  for y in vecteur:
    if x==y:
      counter+=1;
  return counter;

liste=["maison","toto","tata","histoire","maison","royaume","maison"]
print(count_x(liste,"maison"))
```

Est-ce utile de commenter?

+ Langage très lisible

C *Commentaires + qu'indispensables!!*

```c
struct node
{
  char *data;
  struct node *next;
};

void next_list(void *x)
{
  struct node **n=(struct node**)x;
  *n=(*n)->next;
}
void* get_data_list(void* x)
{
  struct node *n=(struct node*)x;
  return n->data;
}

int count_x(void *conteneur,void *valeur,int size_valeur,
            void(*generic_increment)(void*),
            void* (*generic_get_data)(void*))
{
  int compteur=0;
  while(conteneur!=NULL)
    {
      if(memcmp(valeur,generic_get_data(conteneur),size_valeur)==0)
    compteur++;
      generic_increment(&conteneur);
    }
  return compteur;
}
```

```c
int main(int argc,char *argv[])
{
  char *a[]={"maison","toto","tata","histoire","maison","royaume","maison"};
  struct node list[7];
  struct node *head_list=&list[0];
  int k=0;
  int count=0;
  void (*generic_increment)(void *)=next_list;
  void* (*generic_get_data)(void *)=get_data_list;

  for(k=0;k<7;++k)
    {
      list[k].data=a[k];
      if(k<6)
    list[k].next=list+k+1;
      else
    list[k].next=NULL;
    }

  count=count_x(head_list,"maison",6,generic_increment,generic_get_data);
  printf("%d\n",count);

  return 0;
}
```

+ danger mémoire "run-time"

...

Python: Les + / -

+ Langage très lisible

+ Algorithme proche du langage *(apprentissage)*

+ Utilisé en industrie — *script*, *calcul*

- Pas d'apprentissage "hardware"/OS

- Délicat pour code très volumineux

typage dynamique

Python VS

C	+ Aisance codage, clareté - Pas de contrôle bas niveau (embarqué, OS) - Lent
C++	+ Clareté - Lent
Java	+ Simple, moins verbeux + Applicable science (operateurs) - Moins répendu
Matlab	+ Vraie informatique, structures données + Rapide - Moins "sucre syntaxique"

Conditions: si, sinon

Condition *if*

"si"

```
a=5
b=5

if a*b > 22:
  print("j'affiche ce message")
```

(si a fois b est plus grand que 22)

```
a,b=8,9
c=0
if a+b-b*b<=3:
  c=c+3*a

if c/10>c*c:
  print("j'affiche ce message")
```

Condition *if*

"si"

```
a=5
b=5

if a*b > 22:
    print("j'affiche ce message")
```

: début d'un bloc de traitement

espace => bloc d'instructions

Condition *if*

"si"

```
x=12.2

if x>=0 and x<5:
  print("intervalle [0,5[")

if x>2 or x<0:
  print("intervalle [-inf,0[ U ]2;+inf[")
```

Condition *if / else*

"si / sinon"

```
a=5
b=4

if a*b > 22:
  print("j'affiche ce message")
else:
  print("ceci est un autre message")
```

si *a fois b est supérieur à 22*
alors "j'affiche ce message"
sinon
j'affiche "ceci est un autre message"

Condition *if / else*

"si / sinon"

Cas particulier du test d'égalité

```
a=5
if(a==6):
  print("a vaut 6")
else
  print("a ne vaut pas 6")
```

symbole ==
test d'égalité

si *a est égale à 6*
alors affiche "a vaut 6"
sinon
affiche "a ne vaut pas 6"

Math	Code	
$a := b$	a=b	affectation
$a = b$	a==b	test d'égalité *(vaut vrai ou faux)*

L'indentation

```
a,b=1,2
x,y=4,1
if a>5:
    x=x+2
y=y+3

print(y)
```

!=

```
a,b=1,2
x=4
if a>5:
    x=x+2
    y=y+3

print(y)
```

L'indentation fait partie du code!!!

Débat
philosophique
... religieux

```
a,b=1,2
x=4
if a>5:
    x=x+2
     y=y+3

print(x)
```

Ne s'execute pas!

IndentationError: unexpected indent

L'indentation

```
a,b=1,2
x,y=4,1
if a>5:
    x=x+2
y=y+3

print(y)
```

!=

```
a,b=1,2
x=4
if a>5:
    x=x+2
    y=y+3

print(y)
```

\+ Force à la lisibilité du code

Bon pour l'apprentissage

+/- Copié-coller difficile

Mais: copié-coller est à éviter

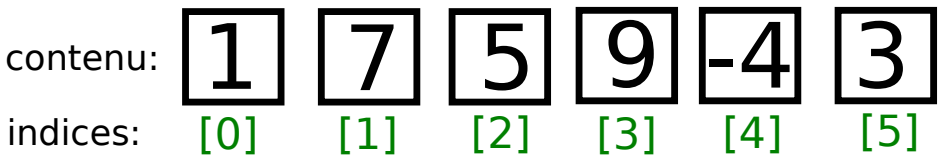
\- Portabilité: compatibilité Tab, espaces, ... (éditeur obligatoire)

Les listes d'éléments

Ensemble éléments: Listes

crochets [...] indiquent une liste

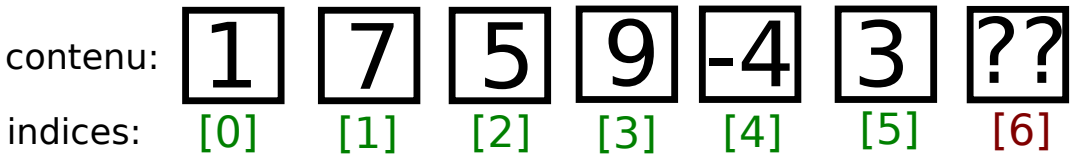
```
vecteur=[1,7,5,9,-4,3]

print(vecteur[0])
print(vecteur[1])
print(vecteur[2])

print(vecteur[0]+vecteur[4])

vecteur[2]=vecteur[4]*vecteur[5]
print(vecteur)
```



Ensemble éléments: Listes

```
vecteur=[1,7,5,9,-4,3]

print(vecteur[6])
```

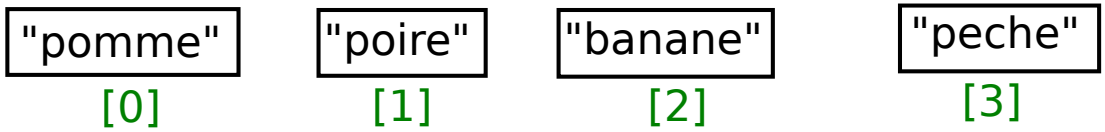
IndexError: list index out of range



Ensemble éléments: Listes

Une liste peut contenir des mots

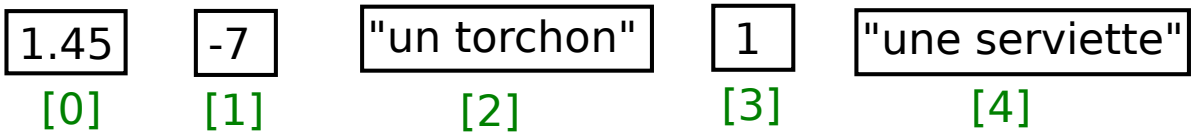
```
vecteur=["pomme","poire","banane","peche"]

vecteur[0]=vecteur[1]+" "+vecteur[2]
print(vecteur)
```



Ensemble éléments: Listes

Une liste peut contenir différents types

```
vecteur_mixte=[1.45,-7,"un torchon",1,"une serviette"]

print(vecteur_mixte[0])
print(vecteur_mixte[2])
print(vecteur_mixte)
```



Ensemble éléments: Listes

Ajouter des éléments dans une liste

```
vec=[4,5,6]
print(vec)
vec.append(7);
vec.append(8);
print(vec)
```

Ensemble éléments: Listes

Supprimer des éléments dans une liste

```
vec=[4,-1,5,7,12]
print(vec)
del(vec[0])
print(vec)
del(vec[2])
print(vec)
```

Ensemble éléments: Listes

Créer une "liste" particulière

```
a=range(4,9)
b=range(8,-2,-3)
print(a[0],a[1],a[2],a[3],a[4])
print(b[0],b[1],b[2],b[3])
```

a | 4 5 6 7 8 |

b | 8 5 2 -1 |

range(debut,fin,*[increment]*)

stop 1 élément avant fin

Ensemble éléments: Listes

Nombre d'éléments d'une liste

```
vec1=[1,4,8,9]
vec2=["canard",7.45,"poireaux"]

longueur_1=len(vec1)
print(longueur_1)

print(len(vec2))
```

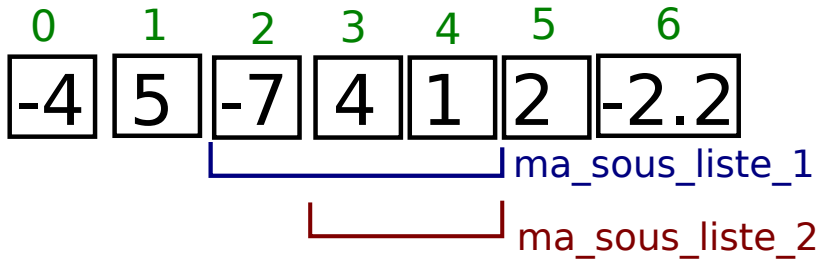
Ensemble éléments: Listes

Indexation inverse

```
vec=[1,-4,7,2,6,8,3]

print(vec[-1])
print(vec[-2])
```

Sous partie d'une liste

```
ma_liste=[-4,5,-7,4,1,2,-2.2]
ma_sous_liste_1=ma_liste[2:5]
ma_sous_liste_2=ma_liste[3:-2]

print(ma_sous_liste_1)
print(ma_sous_liste_2)
```



Trier une liste

```
ma_liste=[4,1,-7,9,5,12,-3]
ma_liste_triee=sorted(ma_liste)

print(ma_liste_triee)
```

```
ma_liste=["velo","cheval","nenuphar","pilote"]
ma_liste_triee=sorted(ma_liste)

print(ma_liste_triee)
```

```
ma_liste=[4,"cheval",7.8]
sorted(ma_liste)
```

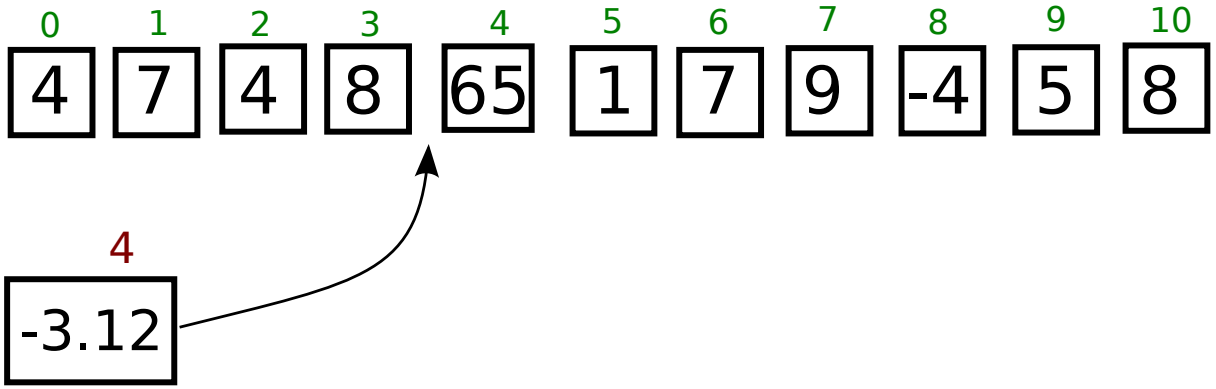
unorderable types str()<int()

Compter nombre d'occurences

```
ma_liste=[7,8,4,-1,4,8,-2,-1,1]
nombre_de_huit=ma_liste.count(8)

print(nombre_de_huit)
```

Insérer un élément dans une liste

```
ma_liste=[4,7,4,8,65,1,7,9,-4,5,8]
ma_liste.insert(4,-3.12)
```



Supprimer par valeur

```
ma_liste=[4,7,4,8,65,1,7,9,-4,5,8]
ma_liste.remove(8)
```

Recherche valeur et supprime l'élément

Ne supprime qu'une valeur (la première trouvée)

Différent de: **del(***ma_liste[k]***)**

supprime le kème élément (indice)

Liste de listes

```
triangle=[[0,0,0],[1,-0.1,1.1],[0,1,0.5]]

print(triangle[0])
print(triangle[1])
print(triangle[2])
print(triangle[1][2])
```

Itération sur les listes

le mot clé "*for*"

Créer des listes

$$L = \{f(k) | k \in [\![0, N[\![\}$$

Exemple pour $f(k) = (k-2)^2$

En *francais*:

L = (k-2)^2 **pour** k variant **dans** [0,N[

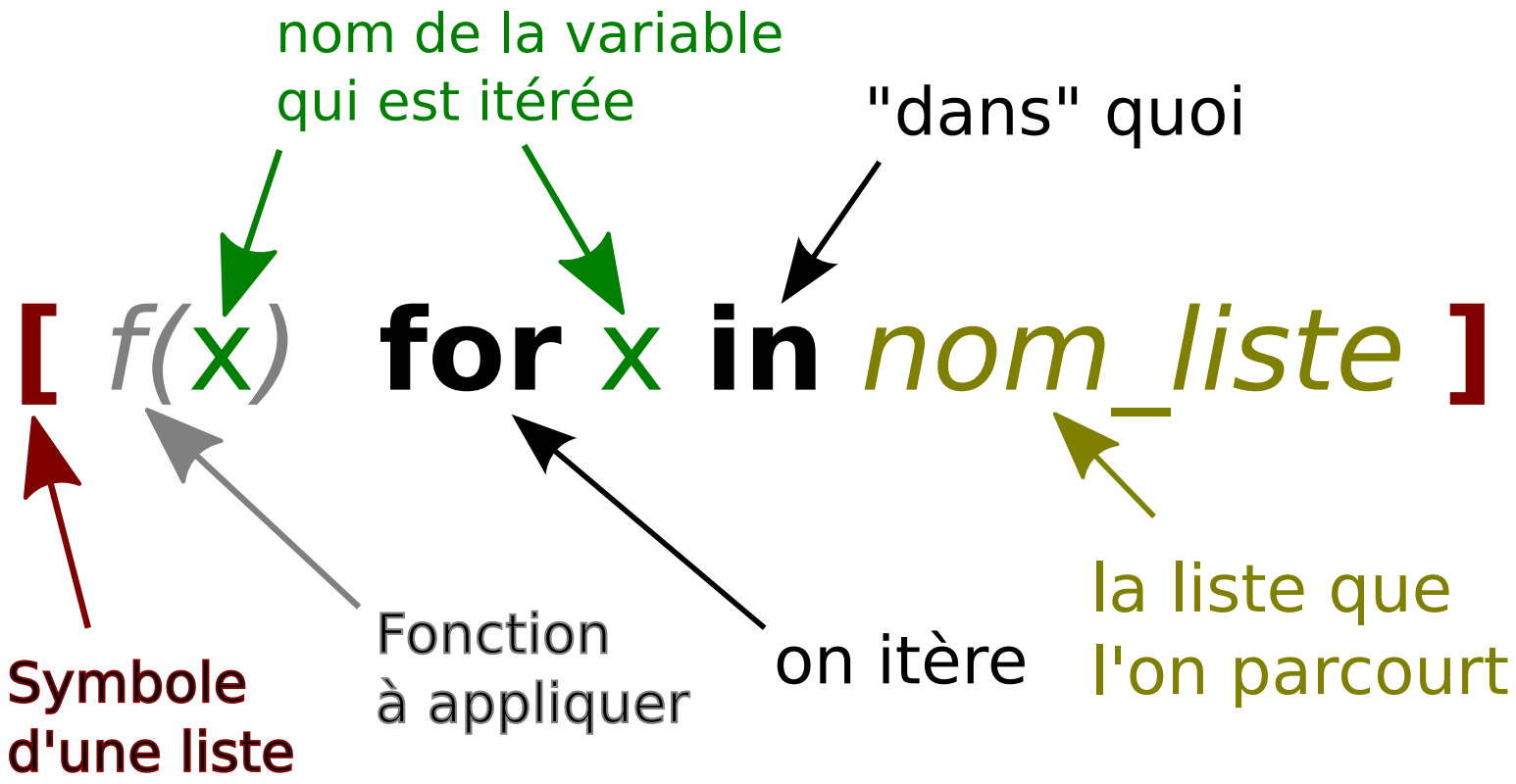
En *code*

```
L=[(k-2)**2 for k in range(0,N)]
```

(** :puissance)

```
N=4
L=[(k-2)**2 for k in range(0,N)]
print(L)
```

La boucle "pour"



Application

Calculer:

$$L = \{a_k \mid k \in [\![-N, N[\![\}$$

$$a_k = k\sqrt{|k-1|}/4$$

(|x| : abs(x))

Application

Calculer:

$$L = \{a_k \mid k \in [\![-N, N[\![\}$$

$$a_k = k\sqrt{|k-1|}/4$$

(|x| : abs(x))

```
L=[(k*abs(k-1)**0.5)/4 for k in range(-N,N)]
```

Application

$$L = \{a_k \mid k \in [\![-N, N[\![\}$$

$$a_k = k\sqrt{|k-1|}/4$$

```
import matplotlib.pyplot as plt

L=[(k*abs(k-1)**0.5)/4 for k in range(-N,N)]

plt.plot(L)
plt.show()
```

affichage (plot=dessine, show=montre à l'écran)

Application

$$L = \{a_k \mid k \in [\![-N, N[\![\}$$

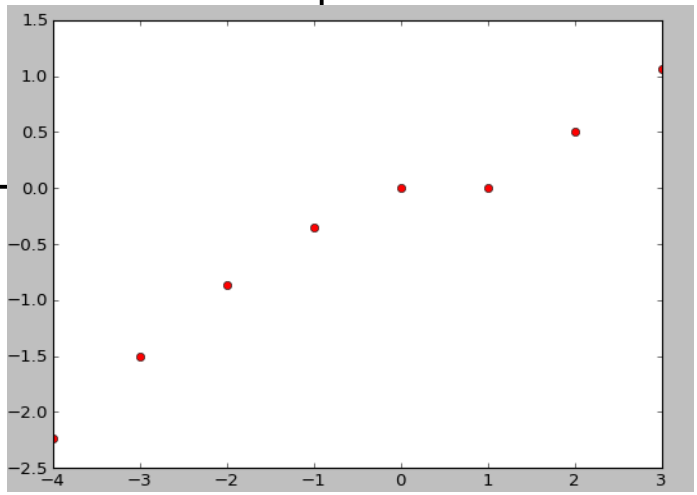
$$a_k = k\sqrt{|k-1|}/4$$

Avec les bonnes abscisses + echantillons

```
import matplotlib.pyplot as plt

L=[(k*abs(k-1)**0.5)/4 for k in range(-N,N)]
abscisses=range(-N,N)

plt.plot(abscisses,L,"ro")
plt.show()
```

en rouge

dessiner des ●



Les fonctions

Les fonctions

Remarque:

$$L = \{f(k) \mid k \in [\![k_0, k_N[\![\}$$

$$f(k) = \ldots$$

```
L=[(k*abs(k-1)**0.5)/4 for k in range(-N,N)]
```

compliqué ... peu lisible!

On souhaiterait écrire:

```
L=[f(k) for k in range(k0,kn)]
```

avec f(k)= ...

Les fonctions

$$L = \{f(k) \mid k \in [\![k_0, k_N[\![\}$$

$$f(k) = k\sqrt{|k-1|}/4$$

```
def f(k):
  return k*abs(k-1)**0.5/4
```

```
k0=-4
kn=8
L=[f(k) for k in range(k0,kn)]
```

Les fonctions

def nom_fonction(argument) :

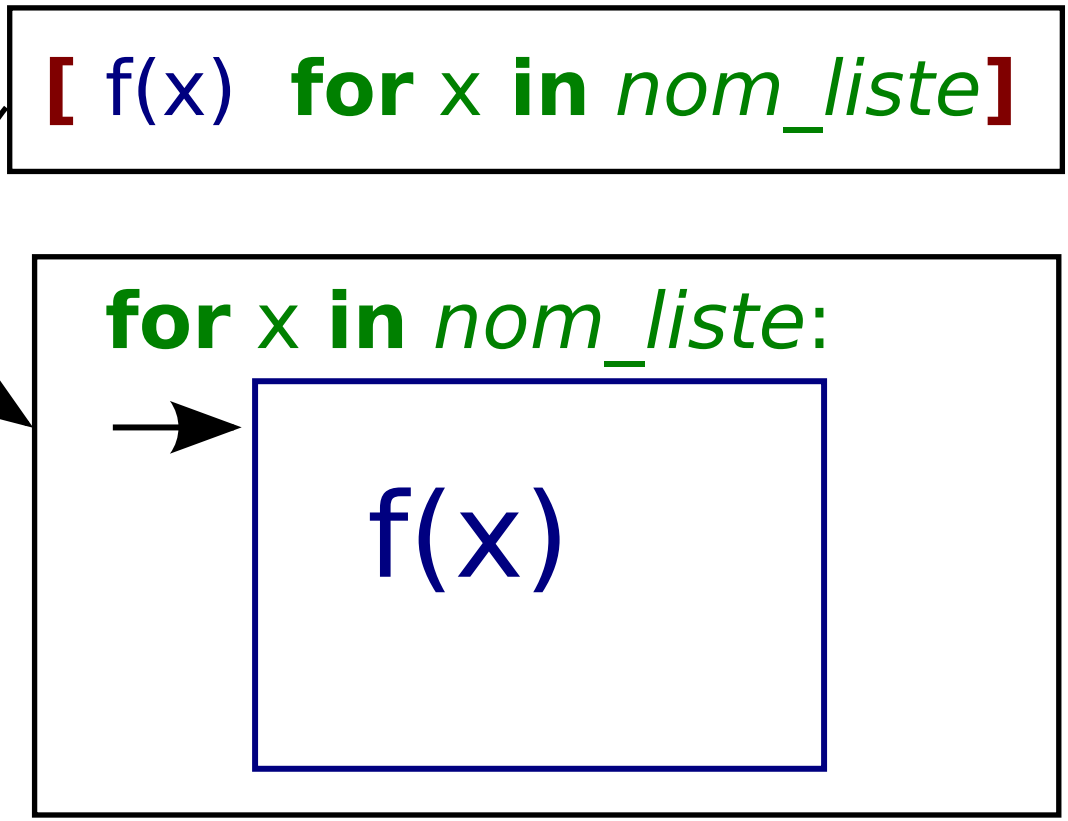
→ *Faire quelque chose ...*

return valeur

```
def f(k):
  return k*abs(k-1)**0.5/4


k0=-4
kn=8
L=[f(k) for k in range(k0,kn)]
```

Boucle "classique"

Remarque: Parfois/souvent f est
- complexe
- ne retourne rien / modifie x (la liste)
- n'est écrite qu'une seule fois



Boucle "classique"

Exemple

```
for x in range(0,8):
  print(x)
```

```
for x in range(-3,4):
  if(x%2 == 0):
    print(x,"est pair")
  else:
    print(x,"est impair")
```

a%b
reste de la division euclidienne

Librairie mathématique et affichage

Numpy: Array

```
import numpy as np

va=np.array([1,4,8,9])
vb=np.array([-4.1,2.2,1,-5])

vc=va+vb

print(vc)
```

array ressemble aux listes
spécialisé pour les nombres

Numpy: Array

```
va=np.array([1,4,8,9])
vb=np.array([-4.1,2.2,1,-5])
```

```
va+vb
va-vb
2*va
vb*4
np.vdot(va,vb)
```

addition
soustraction
multiplication par un scalaire
produit scalaire
...

Rem. On ne mélange pas "mot" et nombre dans un array

Linspace

```
a,b=1.1,4.8
N=8

x=np.linspace(a,b,N)

print(x)
```

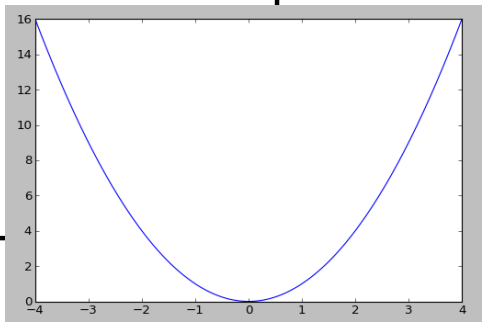
Vecteur uniformément réparti entre [a,b] avec N échantillons

Affichage

Combinaison array + affichage

```
import numpy as np
import matplotlib.pyplot as plt

a,b=-4,4
N=200

x=np.linspace(a,b,N)
y=x**2

plt.plot(x,y)
plt.show()
```

élève au carré élément à élément



Array-range

arange: similaire à range

```
v=np.arange(1,8,2)
print(v)
```

```
1 3 5 7
```

Slicing

```
print(a)
print(a[2:4])
print(a[2:])
print(a[:5])
print(a[1:6:2])
print(a[::3])
```

Vecteurs particuliers

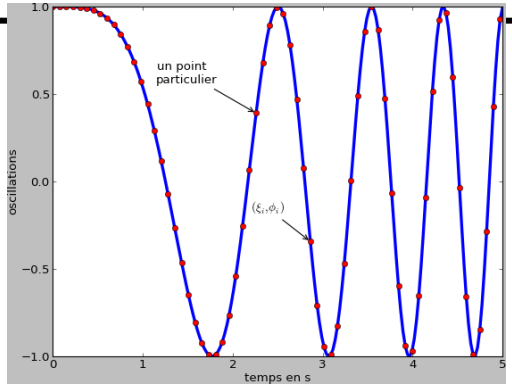
```
va=np.zeros(5)
vb=np.ones(6)
```

va | 0 0 0 0 0

vb | 1 1 1 1 1 1

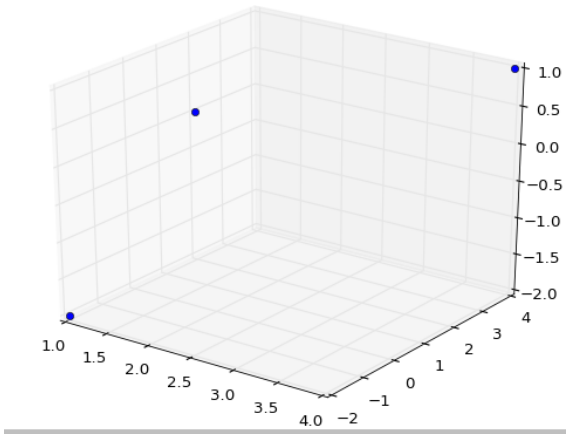
Embellissement graphique

```
N=200
x=np.linspace(0,5,N)
y=np.cos(x*x)



plt.plot(x,y,linewidth=3)
plt.plot(x[::3],y[::3],"ro")
plt.xlabel("temps en s")
plt.ylabel("oscillations")

plt.annotate('un point \nparticulier', xy=(x[90], y[90]),  xycoords='data',
                xytext=(-100, 30), textcoords='offset points',
                arrowprops=dict(arrowstyle="->")
                )

plt.annotate(u'$(\\xi_i,\\phi_i)$', xy=(x[114], y[114]),  xycoords='data',
                xytext=(-60, 30), textcoords='offset points',
                arrowprops=dict(arrowstyle="->")
                )
plt.show()
```



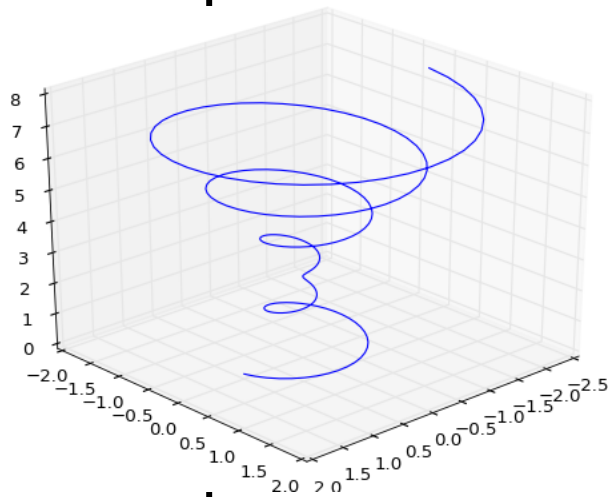
Courbe et points 3D

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

p0=np.array([1,2,0])
p1=np.array([4,4,1])
p2=np.array([1,-2,-2])

triangle=np.array([p0,p1,p2])

x=triangle[:,0]
y=triangle[:,1]
z=triangle[:,2]

print(x,y,z)

fig=plt.figure()
axes3d=fig.gca(projection='3d')
axes3d.plot(triangle[:,0],triangle[:,1],triangle[:,2],"o")
plt.show()
```



Courbe et points 3D

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

N=200
t=np.linspace(0,5*np.pi,N)

x=(1-t/5)*np.cos(2*t)
y=(1-t/5)*np.sin(2*t)
z=t/2

p=np.array([x,y,z])

fig=plt.figure()
axes3d=fig.gca(projection='3d')
axes3d.plot(p[0,:],p[1,:],p[2,:],"-")
plt.show()
```



Matrices

Matrices

```
import numpy as np

A=np.matrix([[1,2,3],[4,5,6]])

print(A)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Matrices

Produit matriciel

```
import numpy as np

A=np.matrix([[1,2,3],
             [4,5,6]])

B=np.matrix([[1,4],
             [1,2],
             [3,10]])

C=A*B

print(C)
```

Matrices

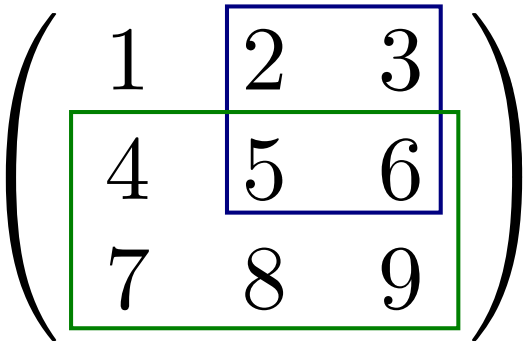
Matrices carrées

```
A=np.matrix([[1,2,3],
             [4,5,6],
             [7,8,9]])

B=np.matrix([[1,4,2],
             [1,2,2],
             [3,-1,1]])

print(A+B)
print(A-B)
print(A*B)
print(A**2)
```

Matrices

Bloc/slicing

```
A=np.matrix([[1,2,3],
             [4,5,6],
             [7,8,9]])


A[0:2,1:3]
A[1:3,:]
```



Operateurs matriciels

```
A=np.matrix([[1,2],
             [4,5]])

np.linalg.det(A)
np.trace(A)
np.linalg.inv(A)
```

linalg = linear algebra

Algèbre linéaire

```
A=np.matrix([[1,2,5],
             [4,5,9],
             [7,8,4]])

b=np.array([4,8,9])

x=np.linalg.solve(A,b)
```

$$\mathrm{A}\,x = b$$

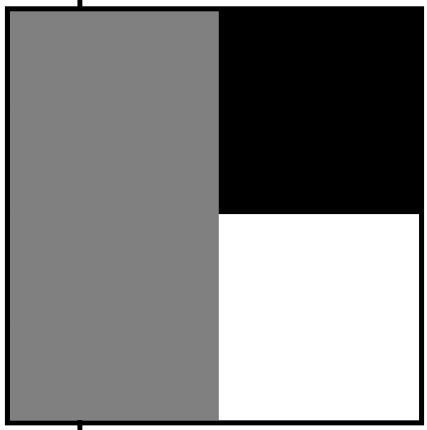
Diagonalisation

```
A=np.matrix([[1,2,3],
             [4,7,8],
             [4,7,1]])

w,P=np.linalg.eig(A)

print(w)
print(P)

print(P*np.diag(w)*np.linalg.inv(P))
```

Affichage matrices/images

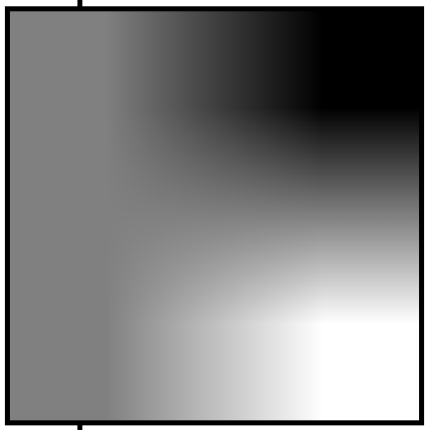
Affichage matrices/images

```
import numpy as np
import matplotlib.pyplot as plt

A=np.matrix([[1,0],
             [1,2]])

im=plt.imshow(A)

im.set_cmap('gray')
im.set_interpolation('nearest')
plt.show()
```



Affichage matrices/images

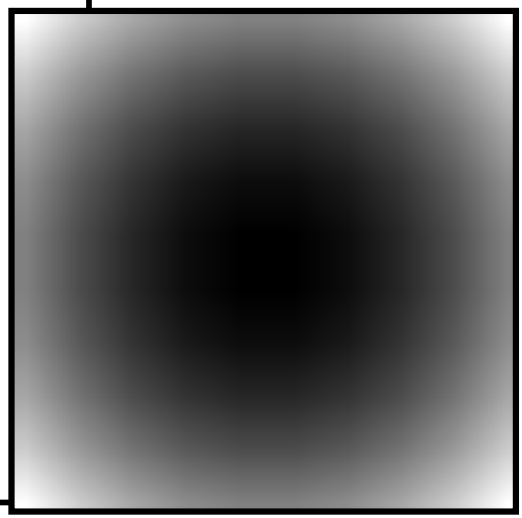
```
import numpy as np
import matplotlib.pyplot as plt

A=np.matrix([[1,0],
             [1,2]])

im=plt.imshow(A)

im.set_cmap('gray')
im.set_interpolation('nearest')
plt.show()
```



Affichage fonctions 2D

```
import numpy as np
import matplotlib.pyplot as plt

N=10
vx=np.linspace(-1.0,1.0,N)
vy=vx

z=np.zeros([N,N])
for kx,x in enumerate(vx):
  for ky,y in enumerate(vy):
    z[kx,ky]=x**2+y**2

plt.imshow(z)
plt.set_cmap("gray")
plt.show()
```

$$f(x,y) = x^2 + y^2$$



Affichage fonctions 2D

Soit:

$$\begin{cases} f(x,y) = \cos\left(h(x,y-2)\,h(x,y+2)\right) \\ h(x,y) = 2\sqrt{x^2+y^2} \end{cases}$$

Afficher f (utiliser la colormap "hot")

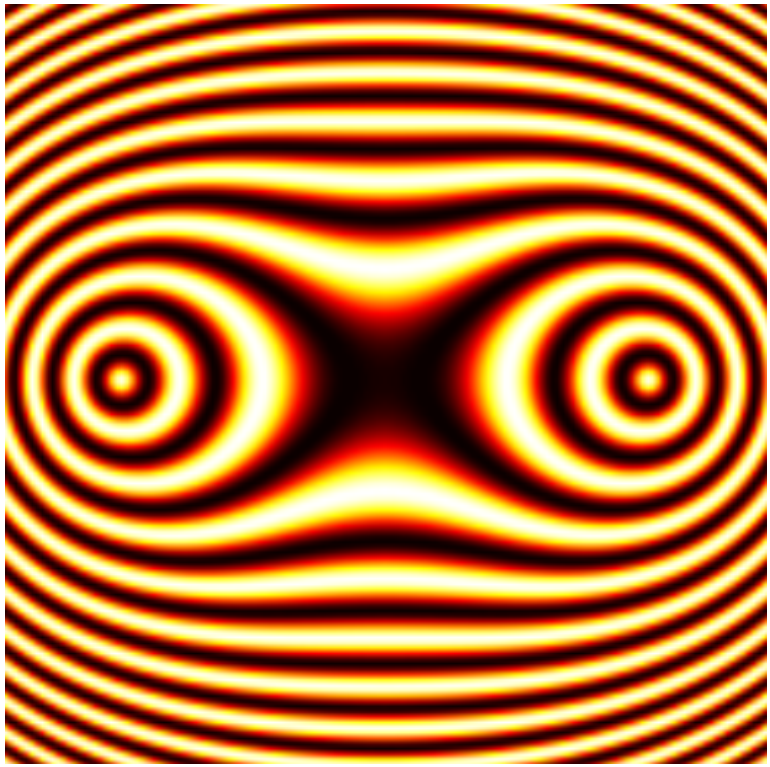
Affichage fonctions 2D

Soit:

$$\begin{cases} f(x,y) = \cos\left(h(x,y-2)\,h(x,y+2)\right) \\ h(x,y) = 2\sqrt{x^2+y^2} \end{cases}$$

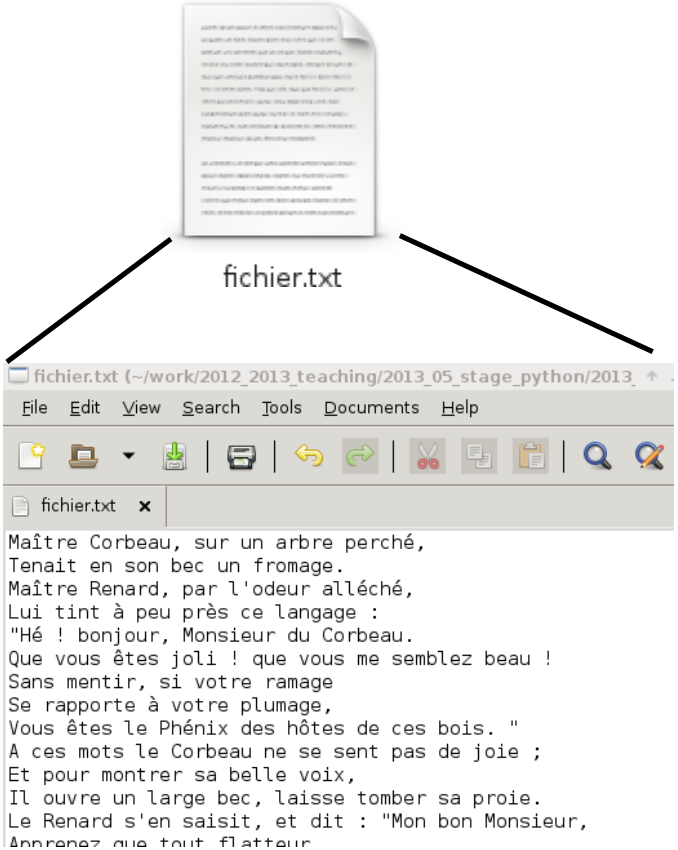
Afficher f (utiliser la colormap "hot")

```
def h(x,y):
  return 2*(x**2+y**2)**(0.5)

def f(x,y):
  z=np.cos(h(x,y-2)*h(x,y+2))
  return z

N=150
vx,vy=np.linspace(-3,3,N),np.linspace(-3,3,N)

z=np.zeros([N,N])

for kx,x in enumerate(vx):
  for ky,y in enumerate(vy):
    z[kx,ky]=f(x,y)

print(z)
im=plt.imshow(z)
im.set_cmap("hot")
plt.show()
```



Fichiers

Lecture d'un fichier



Lecture ligne à ligne

```
fichier=open("fichier.txt")

for ligne in fichier:
  print(ligne)

fichier.close()
```

Traitement d'un texte

```
fichier=open("fichier.txt")

for ligne in fichier:
  nouvelle_phrase=ligne.replace("Corbeau","Pelican")
  print(nouvelle_phrase)

fichier.close()
```

Maître Pelican, sur un arbre perché,
Tenait en son bec un fromage.
Maître Renard, par l'odeur alléché,
Lui tint à peu près ce langage :
"Hé ! bonjour, Monsieur du Pelican.
...

Ecriture fichier

write/ecrit
(supprime/créé un fichier vierge)

```
fichier=open("mon_fichier.txt","w")

fichier.write("Bonjour fichier \n")
fichier.write("3+3="+str(3+3))

fichier.close()
```

```
Bonjour fichier
3+3=6
```