

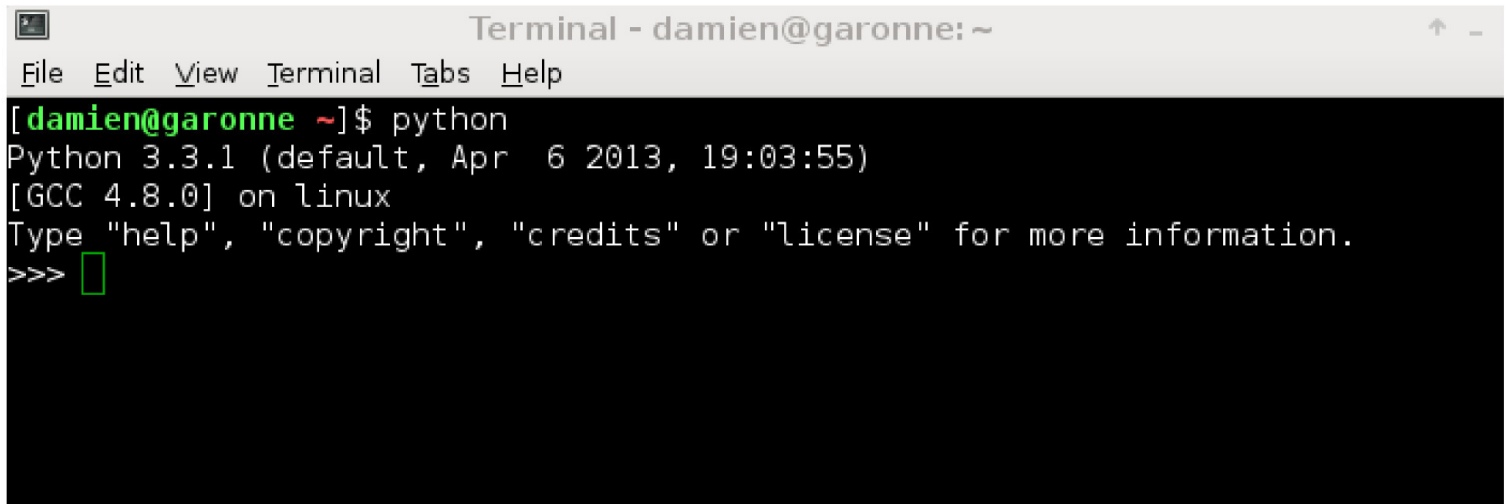
Introduction à la programmation

Python



Damien Rohmer

Premier "programme"



```
Terminal - damien@garonne: ~
File Edit View Terminal Tabs Help
[damien@garonne ~]$ python
Python 3.3.1 (default, Apr  6 2013, 19:03:55)
[GCC 4.8.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Premier "programme"

```
Terminal - damien@garonne: ~  
File Edit View Terminal Tabs Help  
[damien@garonne ~]$ python  
Python 3.3.1 (default, Apr 6 2013, 19:03:55)  
[GCC 4.8.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 7+8  
15  
>>> 
```

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=5` ← a est un nombre (entier)
`b=4.12` ← b est un nombre (à virgule)
`c="du texte"` ← c est un texte

`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:

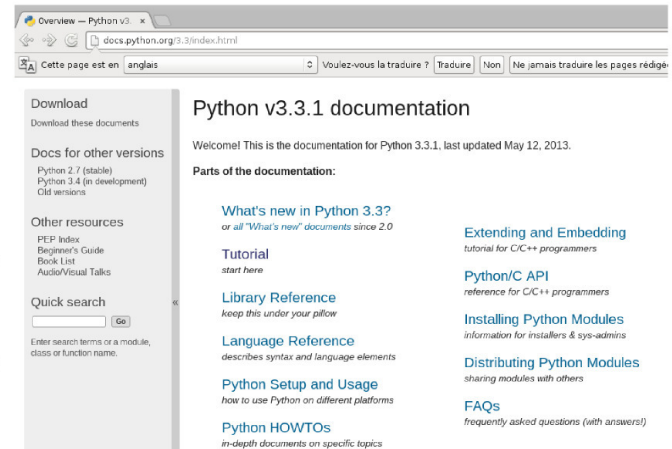
help(*nom_fonction*)

ex. help(pow)

Site web:

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

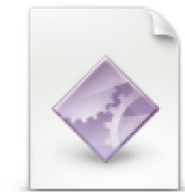
<http://docs.python.org/3/index.html>



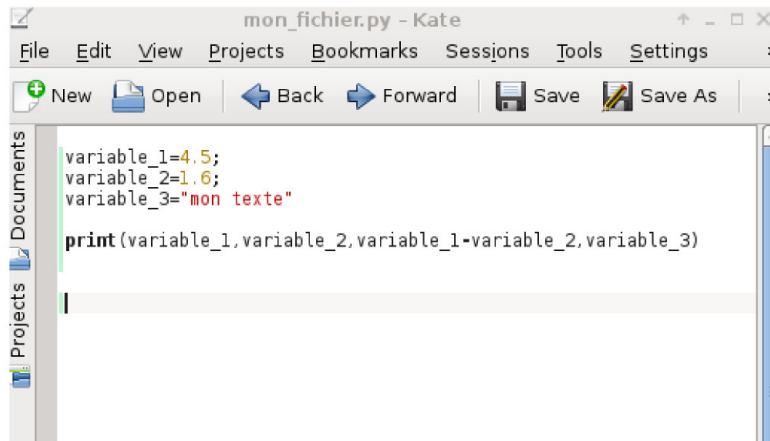
Ecriture dans un fichier

Ecrire ligne à ligne est fastidieux ...

On écrit d'abord dans un fichier texte



mon_fichier.py



```
mon_fichier.py - Kate
File Edit View Projects Bookmarks Sessions Tools Settings
New Open Back Forward Save Save As
Documents
variable_1=4.5;
variable_2=1.6;
variable_3="mon texte"
print(variable_1,variable_2,variable_1-variable_2,variable_3)
```

(.py = fichier texte lisible par Python)

On lance Python sur le fichier

```
[damien@damien_pc ~/work/2012_2013_teaching
python/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 défaut, pyscripter

Editeur type Matlab: **Spyder**

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script with the following content:

```
1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This temporary script file is located here:
6 /home/damien/.spyder2/.temp.py
7 """
8
9 a=5
10 b=[4,5,8,6]
11 print("Hello world "+str(a))
12
13
```

The Variable explorer on the right shows the following variables:

Name	Type	Size	Value
a	int	1	5
b	list	3	<list @ 0x2854800>

The Console at the bottom shows the execution of the script, resulting in the following output:

```
File ~/home/damien/Downloads/quiver demo (1).py", line 11, in <module>
  from pylab import *
ImportError: No module named pylab
>>> runfile(r'~/home/damien/.spyder2/.temp.py', wdir=r'~/home/damien/.spyder2')
Hello world 5
>>> runfile(r'~/home/damien/.spyder2/.temp.py', wdir=r'~/home/damien/.spyder2')
Hello world 5
>>> runfile(r'~/home/damien/.spyder2/.temp.py', wdir=r'~/home/damien/.spyder2')
Hello world 5
>>>
```

A small dialog box titled "b - List (3 elements)" is also visible, showing the contents of variable 'b':

idx	Type	Size	Value
0	int	1	4
1	float	1	5.8
2	int	1	6

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

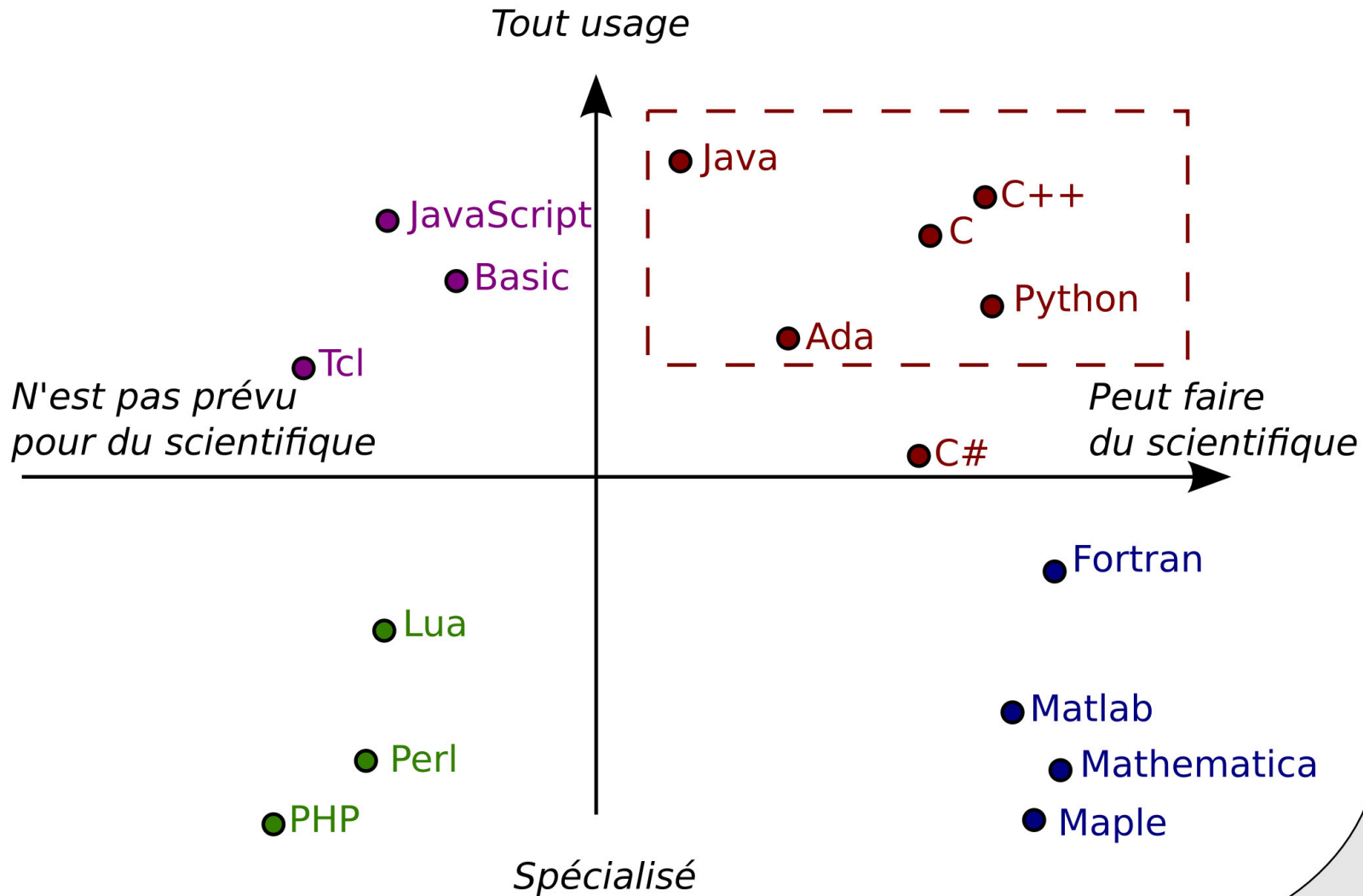
Python 3.0 en 2009



Python devient un acteur majeur du monde du calcul scientifique

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

Python: positionnement



Python: positionnement

Simple, Lisible



● Python

● Java

● Ada

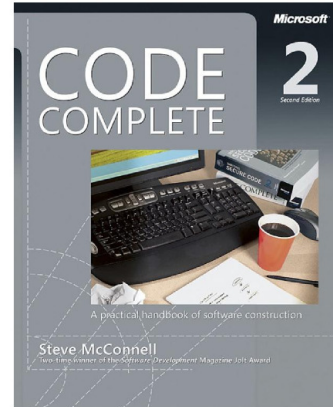
● C

● C++

Complexe

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

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

Python

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

*Ce qui est simple
s'écrit simplement*

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 (opérateurs)
- Moins répandu

Matlab

- + Vraie informatique, structures données
- + Rapide
- Moins "sucre syntaxique"

Conditions: si, sinon

Condition *if*

"si"

```
a=5
```

```
b=5
```

```
if a*b > 22:                (si a fois b est plus grand que 22)  
    print("j'affiche ce message")
```

```
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
```

: début d'un bloc de traitement

```
if a*b > 22:
```

```
    print("j'affiche ce message")
```

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

$a := b$

$a = b$

Code

$a=b$ affectation

$a==b$ test d'égalité
(vaut *vrai* ou *faux*)

Condition *if / else*

"si / sinon"

Conditions if/elif/else

elif = else, if (/ sinon, si ...)

```
if x>=0 and x<2:  
    print("intervalle [0,2[" )  
elif x<4:  
    print("intervalle ]-inf,0[ U [2,4[" )  
elif x<=5:  
    print("intervalle [4,5]      ")  
else:  
    print("intervalle ]5,+inf[" )
```

Condition *if* imbriqués

Quelle courbe dessine y si x varie ?

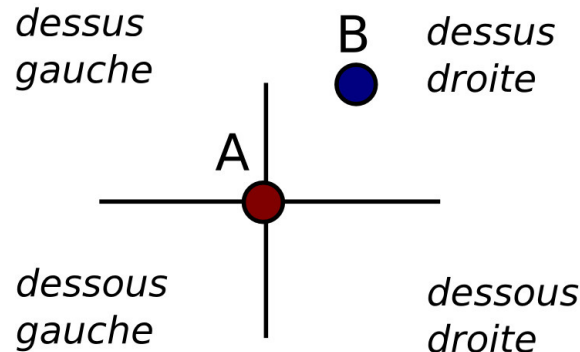
```
if x >= 0:  
    if x < 1:  
        y = x * x  
    else:  
        y = 2x - 1  
else:  
    if x > -1:  
        y = -x * x  
    else:  
        y = 2x + 1
```

Condition *if* imbriqués

Application:

Soit 2 points $A=(x_1,y_1)$ et $B=(x_2,y_2)$

Indiquer si A est au | dessus/dessous de B
| gauche/droite



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'exécute 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)
```

contenu:

1	7	5	9	-4	3
---	---	---	---	----	---

indices:

[0]	[1]	[2]	[3]	[4]	[5]
-----	-----	-----	-----	-----	-----

Ensemble éléments: Listes

```
vecteur=[1,7,5,9,-4,3]  
print(vecteur[6])
```

IndexError: list index out of range

contenu:	1	7	5	9	-4	3	??
indices:	[0]	[1]	[2]	[3]	[4]	[5]	[6]

Ensemble éléments: Listes

Une liste peut contenir des mots

```
vecteur=["pomme", "poire", "banane", "peche"]  
vecteur[0]=vecteur[1]+" "+vecteur[2]  
print(vecteur)
```

"pomme"

[0]

"poire"

[1]

"banane"

[2]

"peche"

[3]

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)
```

1.45

[0]

-7

[1]

"un torchon"

[2]

1

[3]

"une serviette"

[4]

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]
```

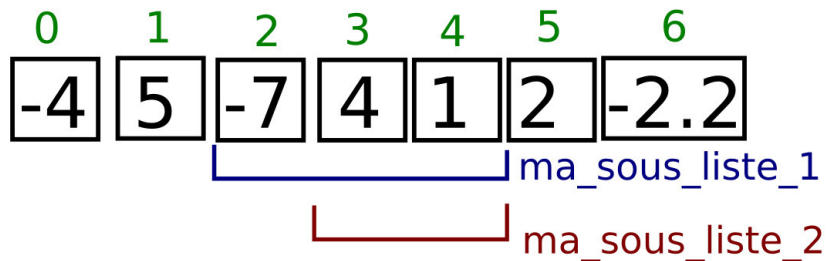
```
print(vec[-1])
```

```
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)
```

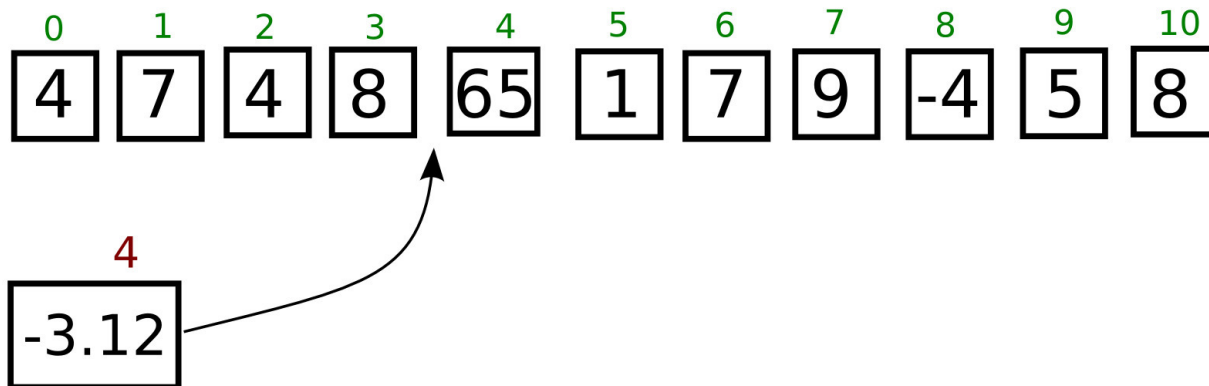
*unorderable types
str() < int()*

Compter nombre d'occurrences

```
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])
```

Egalité de liste

L'affectation de liste référence la même entité

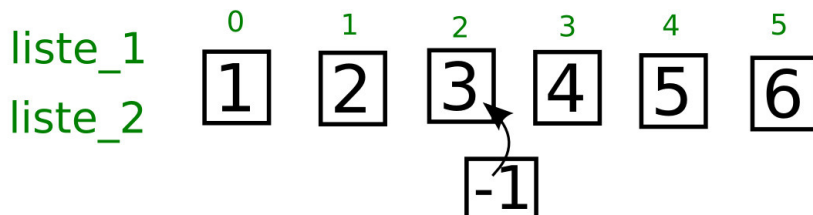
```
liste_1=[1,2,3,4,5,6]
liste_2=["a","b","c","d"]

liste_2=liste_1

print(liste_2)

liste_2[3]=-1

print(liste_2)
print(liste_1)
```



Copie de liste

Si l'on souhaite dupliquer une liste, on appelle explicitement `list(nom_liste)`

```
liste_1=[1,2,3,4,5,6]
liste_2=["a","b","c","d"]

liste_2=list(liste_1)

print(liste_2)

liste_2[3]=-1

print(liste_2)
print(liste_1)
```

création
d'une copie de la liste

	0	1	2	3	4	5
liste_1	1	2	3	4	5	6
	0	1	2	3	4	5
liste_2	1	2	3	-1	5	6

Copie de liste

```
liste_1=[1,4,7,8,5,1,4,7]
```

```
sous_liste=liste_1[2:6]
```

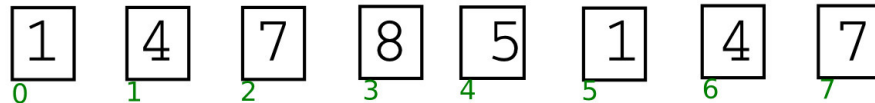
```
sous_liste[2]=15
```

```
print(sous_liste)
```

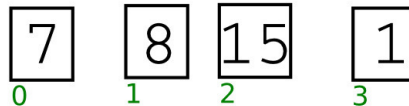
```
print(liste_1)
```

Copie "automatique"
par sous liste

liste_1:



sous_liste:



Itération sur les listes

le mot clé "*for*"

Créer des listes

$$L = \{f(k) \mid k \in \llbracket 0, N \llbracket \}$$

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

En *français*:

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

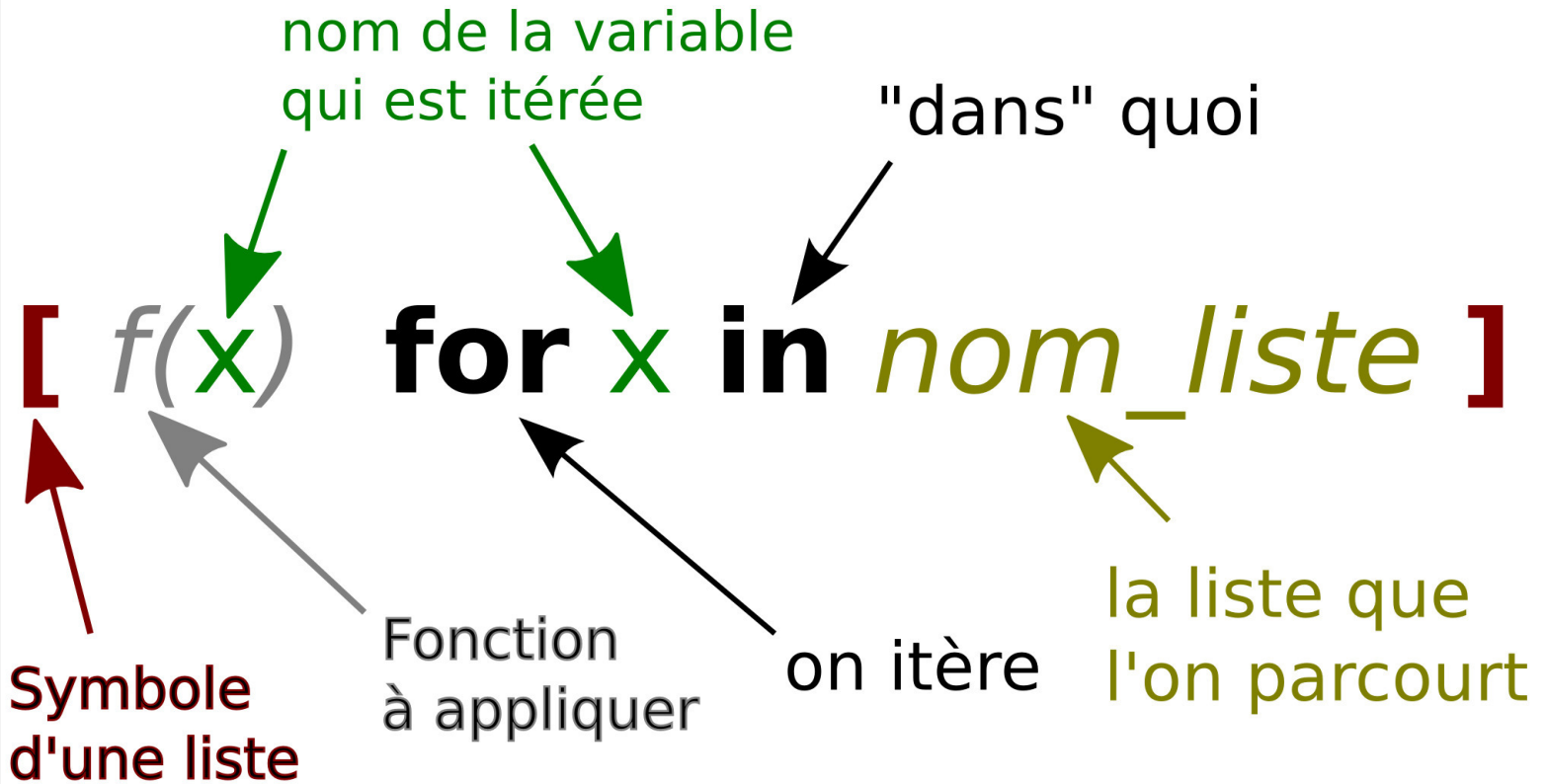
En *code*

```
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

Calculus:

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

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

$(|x| : \text{abs}(x))$

Application

Calculus:

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

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

$(|x| : \text{abs}(x))$

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

Application

$$L = \{a_k \mid k \in \llbracket -N, N \rrbracket\}$$

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

```
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

$$L = \{a_k \mid k \in \llbracket -N, N \rrbracket\}$$

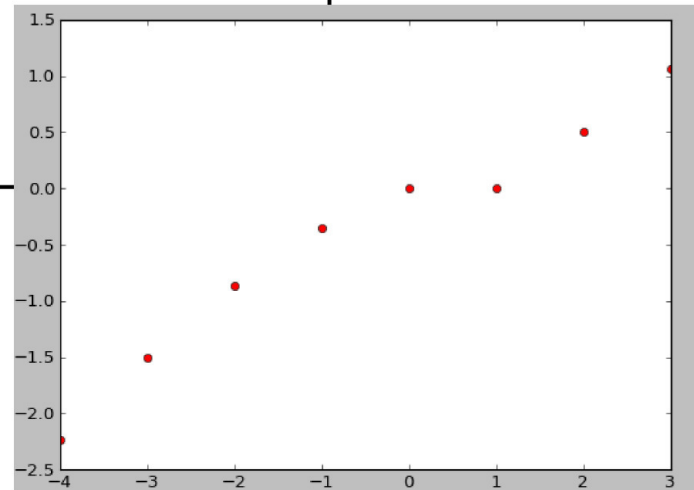
$$a_k = k\sqrt{|k-1|}/4$$

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:

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

$$f(k) = \dots$$

```
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 $f(k) = \dots$

Les fonctions

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

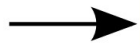
$$f(k) = k\sqrt{|k-1|}/4$$

```
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

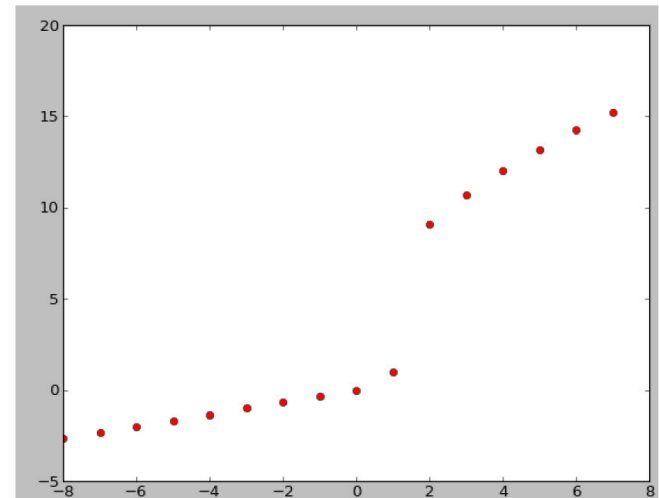
```
def f(k):  
    return k*abs(k-1)**0.5/4  
  
k0=-4  
kn=8  
L=[f(k) for k in range(k0,kn)]
```

Les fonctions

Les fonctions peuvent être *compliquées*

$$\left| \begin{array}{ll} f(k) = 5\sqrt{k} + 2 & k \geq 2 \\ f(k) = k^2 & k \in] - 1, 2[\\ f(k) = k/3 & k \leq -1 \end{array} \right.$$

```
def f(k):  
    if k >= 2 :  
        return 5*k**0.5+2  
    if k <= -1 :  
        return k/3  
    if -1 < k < 2 : #(optionnel)  
        return k**2  
  
L=[f(k) for k in range(-8,8)]  
plt.plot(range(-8,8),L, "ro")  
plt.show()
```



Les fonctions : retour d'arguments

```
def f(x):  
    return [x[0]**2,x[1]**2,"chien"]
```

```
a = f([1,4])  
print(a) retour "liste"
```

```
x0,x1,s = f([1,4])  
print(x0,x1,s) récupère chaque argument  
(unpack)
```

```
b,c = f([1,4])  
print(b,"",c) Erreur  
ValueError: too many values to unpack
```

Multi fichiers

ma_lib.py

```
import math

def norm(x):
    return math.sqrt(x[0]**2+x[1]**2)

def norm3(x):
    return (x[0]**3+x[1]**3)**(1/3.0)
```

```
import ma_lib
```

```
x=[1,8]
n2=ma_lib.norm(x)
n3=ma_lib.norm3(x)

print(n2,n3)
```

namespace automatique
(pas de collisions de noms)

Multi fichiers



ma_lib.py

```
import math

def norm(x):
    return math.sqrt(x[0]**2+x[1]**2)

def norm3(x):
    return (x[0]**3+x[1]**3)**(1/3.0)
```



```
from ma_lib import norm3
```

```
x=[1,8]
#n2=norm(x) non accessible
n3=norm3(x) plus de namespace
print(n3)
```

Multi fichiers

ma_lib.py

```
import math


def norm(x):
    return math.sqrt(x[0]**2+x[1]**2)

def norm3(x):
    return (x[0]**3+x[1]**3)**(1/3.0)
```

```
from ma_lib import *

x=[1,8]
n2=norm(x)    Tout est
n3=norm3(x)   accessible

print(n3)
```

 A éviter

Les fonctions

Les fonctions peuvent prendre des paramètres

$$f(k) = ak^2 + b$$

```
def f(k,a,b):  
    return a*k**2+b
```

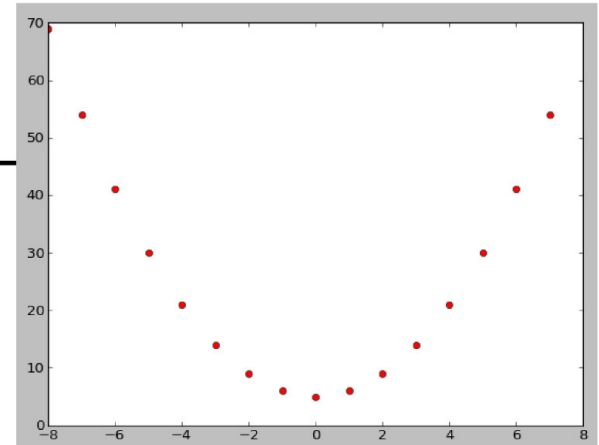
```
a=1
```

```
b=5
```

```
L=[f(k,a,b) for k in range(-8,8)]
```

```
plt.plot(range(-8,8),L,"ro")
```

```
plt.show()
```



Les fonctions

Les fonctions peuvent être vectorielles

 \mathbb{R}^3 \mathbb{R}^2

$$f : (x, y, z) \mapsto (x^2 + y, xy)$$

```
def f(x, y, z):  
    return [x**2+y, x*y]
```

```
v=f(1.5, 2.2, -1.1)  
print(v)
```

Fonctions disponibles

Somme des éléments

$$S = \sum_k x_k$$

```
vec=[1,7,8,4,5]  
S=sum(vec)
```

Application:

$$n = \left(\sum_k x_k^2 \right)^{1/2}$$

```
vec=[1,7,8,4,5]  
n=sum([x_k**2 for x_k in vec])**0.5
```

Min/Max des éléments

```
vec=[1,7,8,4,-2,5]  
a=max(vec)  
b=min(vec)  
print(a)  
print(b)
```

Il existe un élément ...

Soit $(a_k)_{k \in [0, N[}$

Question: $\exists k \in [0, N[, a_k = 4$

En français:

Il existe au moins un élément tel que a_k égale 4
pour k variant entre $[0, N[$

En code:

```
any(x==4 for x in vec)
```

True

(type booléen)

False

Il existe un élément ...

Exemple:

```
vec=[1,7,8,4,-2,5]
est_ce_vrai = any(x==4 for x in vec)

print(est_ce_vrai)
```

```
vec=[1,7,8,4,-2,5]
est_ce_vrai = any(x>9 for x in vec)

print(est_ce_vrai)
```

Tous les éléments ...

Soit $(a_k)_{k \in [0, N[}$

Question: $\forall k \in [0, N[, a_k < 12$

En français:

Tous les a_k sont inférieurs à 12
pour k variant entre $[0, N[$

En code:

```
all(x < 12 for x in vec)
```

True

(type booléen)

False

Boucle for "avancée"

Boucle sur des mots

```
ensemble=["pommes","poires","champignons","poivrons"]  
[print("j'aime manger des "+aliment) for aliment in ensemble]
```

j'aime manger des pommes
j'aime manger des poires
j'aime manger des champignons
j'aime manger des poivrons

Boucle sur plusieurs vecteurs

```
ensemble_matiere=["math","physique","chimie","informatique"]  
ensemble_notes=[12.1,8.4,12.3,7.8]  
  
[print(matiere,":",note) for matiere , note  
in zip(ensemble_matiere,ensemble_notes)]
```

→ met les éléments ensemble

```
math : 12.1  
physique : 8.4  
chimie : 12.3  
informatique : 7.8
```

Boucle "classique"

Remarque: Parfois/souvent f est

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

```
[ f(x) for x in nom_liste ]
```

```
for x in nom_liste:
```

```
    f(x)
```



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

Boucle "classique"

Soit:

```
ensemble_matiere=["math", "physique", "chimie", "informatique"]  
ensemble_notes=[9.1, 8.4, 16.3, 7.8]
```

Afficher "attention + nom_matiere" si note < 10

Afficher "ATTENTION + nom_matiere" si note < 8

Afficher "TB + nom_matiere" si note > 15

Boucle "classique"

Soit:

```
ensemble_matiere=["math", "physique", "chimie", "informatique"]  
ensemble_notes=[9.1,8.4,16.3,7.8]
```

Afficher "attention + nom_matiere" si note<10

Afficher "ATTENTION + nom_matiere" si note<8

Afficher "TB + nom_matiere" si note>15

```
for matiere,note in zip(ensemble_matiere,ensemble_notes):  
  
    if(note<8):  
        print("ATTENTION",matiere,"!!!")  
    elif(note<10):  
        print("attention",matiere)  
    elif(note>15):  
        print("TB",matiere)
```

Boucle "classique"

Modification d'éléments:

Ajouter 2 à toutes les notes si elles sont inférieures à 8

```
notes=[9.1,8.4,16.3,7.8]
for k in range(len(notes)):
    if(notes[k]<8):
        notes[k]=notes[k]+2
```

Récupérer valeur et indice

```
ma_liste=[9.1,8.4,16.3,7.8]
for indice,valeur in enumerate(ma_liste):
    print(indice,valeur)
```

```
0 9.1
1 8.4
2 16.3
3 7.8
```

Similaire à :

```
for indice in range(len(ma_liste)):
    valeur=ma_liste[indice]

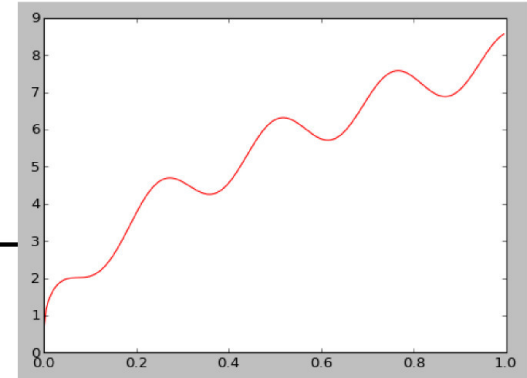
    print(indice,valeur)
```

Récupérer valeur et indice

Application: Rendre une suite croissante

Soit la suite $(a_k)_{k \in \llbracket 0, N \llbracket$

Si $a_{k+1} < a_k$, alors $a_{k+1} := a_k$



$$\forall k \in \llbracket 0, N \llbracket, a_k := f(k/N)$$

$$\forall x \in [0, 1], f(x) := 8\sqrt{x} + 0.6 \cos(25x)$$

Récupérer valeur et indice

Application: Rendre une suite croissante

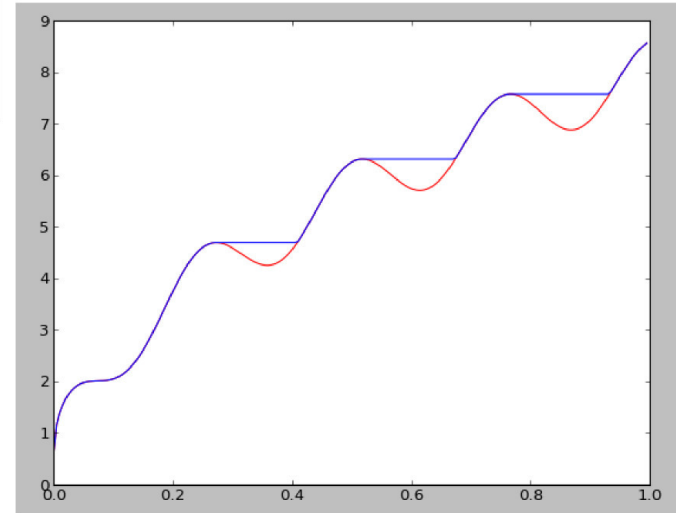
Soit la suite $(a_k)_{k \in \llbracket 0, N \llbracket$

Si $a_{k+1} < a_k$, alors $a_{k+1} := a_k$

$\forall k \in \llbracket 0, N \llbracket, a_k := f(k/N)$

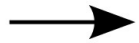
$\forall x \in [0, 1], f(x) := 8\sqrt{x} + 0.6 \cos(25x)$

```
for k, a_k in enumerate(a):  
    if k+1 < N and a[k+1] < a[k]:  
        a[k+1] = a[k]
```



Boucle while / "tant que"

while *condition_vraie* :



Faire quelque chose

```
a=5
while a>2:
    a=0.65*(a+1)
    print(a)
```

On ne connaît pas forcément le nombre d'itérations

Arguments en Python

Typage fort

Python est un langage "fortement" typé
(contrairement au C !)

```
def func(a):  
    s=type(a)  
    print(s)  
  
    return [a,5]
```

```
func("cheval")  
func(8)  
func(1.4)  
func(True)  
func([4,7])  
func([])
```

Le type est connu
à tout instant

```
<class 'str'>  
<class 'int'>  
<class 'float'>  
<class 'bool'>  
<class 'list'>  
<class 'list'>
```


Type

```
def func(a):  
    if isinstance(a, str):  
        return "string"  
    elif isinstance(a, float):  
        print(a+4)  
    else:  
        return [4,5]
```

```
func("cheval")  
func(4.5)
```

rem.
Retour variable



isinstance : comparaison du type

ID

```
a=[4,7,8];  
b=[4,7,8];  
c=a;  
  
print(id(a),id(b),id(c))
```

id ~ adresse

=> Mais on n'accède pas à l'objet à partir de l'id!

Fonctions arguments

Les fonctions peuvent être passées en tant qu'argument

```
def affichage_simple(T):  
    print(T)  
  
def affichage_debug(T):  
    print("type : ", type(T))  
    print("id : ", id(T))  
    print("length : ", len(T))  
    print(T)  
  
def squared(v, afficher):  
    s=0  
    v2=[vk**2 for vk in v]  
    afficher(v2)  
  
v=range(-5, 15, 4)  
aff=affichage_debug  
  
squared(v, aff)
```

variable →

ex.

Optimisation/
Intégration
numérique