SEANCE 3

Fonctions et graphiques, discrétisation

22 octobre 2025

Fonctions

Fonction : une suite d'instructions qui permet de faire réaliser à l'ordinateur une tâche bien spécifique. Elles permettent

- de segmenter le problème global en plusieurs sous-tâches simples
- d'augmenter la clarté du programme, la rapidité pour le débugguer et le développer vers des besoins plus complexes.

Structure générale :

```
def ma_fonction(arguments):
    ...
    instructions
    ...
    return resultat
```

Fonction (suite)

- Le mot clé qui introduit la définition de la fonction est **def**, suivi du nom de la fonction complété par des **parenthèses** et le symbole **double points**.
- Les parenthèses contiennent les **arguments** de la fonction, mais peuvent parfaitement rester vides si la fonction ne nécessite pas d'arguments.
- Puis les instructions sont obligatoirement écrites avec une **indentation**.
- L'instruction **return** renvoie le résultat de la fonction, mais n'est pas obligatoire.

Une fois définie, la fonction peut être exécutée normalement comme une simple ligne d'instruction.

```
>>> arg = ...
>>> ma_fonction(arg)
resultat
```

Fonction à une variable

Exemple 1 : définissons une fonction parabolique $f(x) = 2x^2 + 3x + 4$, et évaluons-la en $x = -2.45$:

```
def parabole(x):
    f = 2*x*x+3*x+4
    return f
>>> parabole(-2.45)
8.655
```

Exemple 2 : Le mot clé **return** n'est pas obligatoire. Le résultat peut être simplement écrit à l'écran avec l'instruction **print**.

```
def conversion_hms(secondes):
    h = secondes // (3600)
    m = (secondes  - h*3600) // 60
    s = secondes - h*3600 - m*60
    print(h,' heures ',m,' minutes ',s,' secondes')
>>> conversion_hms(12345)
3  heures  25  minutes  45  secondes
```

Fonction à une variable (suite)

Des variables peuvent être définies à l'intérieur d'une fonction. Cependant elles ne seront définies que **localement**, c'est-à-dire que l'ordinateur ne connaît la valeur de la variable que lors de l'exécution de la fonction.
Exemple 3 : Conversion euro/dollar avec définition du facteur de conversion à l'intérieur de la fonction :

```
def conversion_euro_to_dollar(euros):
    facteur = 1.27
    dollars = euros * facteur
    print(euros,' euros = ',dollars,' dollars')
    return dollars
>>> conversion_euro_to_dollar(30)
30  euros =  38.1  dollars
38.1
>>> print(facteur)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'facteur' is not defined
```

La variable `facteur` est non définie en dehors de sa fonction.

Fonction à une variable (suite)

N'importe quel type de variable peut être argument d'une fonction, comme par exemple une liste :

```
def somme(liste):
    total = 0
    for element in liste :
        total = total + element
    return total

>>> somme([2,-3.5,7.12,99])
104.62
```

Fonction à une variable (suite)

ou une chaîne de caractères :

```
def is_email(chaine):
    if '@' in chaine :
        print(chaine,"est une adresse email")
    else :
        print(chaine,"n'est pas une adresse email")

>>> is_email("www.moncourspython.fr")
www.moncourspython.fr n'est pas une adresse email
>>> is_email("prenom.nom@moncourspython.fr")
prenom.nom@moncourspython.fr est une adresse email
```

Fonction à plusieurs variables

Une fonction peut admettre plusieurs arguments, de types différents, séparés par des virgules.

Exemple 1 : Convertir une heure en seconde :

```
def conversion_secondes(h,m,s):
    secondes = h*3600 + m*60 + s
    print(h,'h',m,'m',s,'s = ',secondes,'secondes')

>>> conversion_secondes(3,25,45)
3 h 25 m 45 s =  12345 secondes
```

Fonction à plusieurs variables (suite)

Certains des arguments peuvent être facultatifs, si on leur donne une **valeur par défaut** avec le signe $=$.
Exemple 2 : Calcul d'un angle de réfraction par la troisième loi de Descartes, souvent le milieu 1 est l'air d'indice $n_1 = 1$.

```
from math import asin,sin
def angle_refraction(theta1,n2,n1=1):
    theta2 = asin( n1*sin(theta1)/n2 )
    return theta2

>>> from math import pi
>>> angle_refraction(pi/4,1.33) # n1=1 par defaut
0.560558413742460 5
>>> angle_refraction(pi/4,1.33,1.5)  # n1=1.5
0.923121570361274 3
```

mais attention, les arguments facultatifs sont toujours placés après les arguments obligatoires !

Fonction à plusieurs variables (suite)

Les arguments facultatifs peuvent être intéressants pour faire passer des variables booléennes qui changent le comportement de la fonction, par exemple si elle doit imprimée ou non le résultat à l'écran :

```
def conversion_euro_to_dollar(euros,verbose=True):
    # verbose signifie verbeux anglais
    facteur = 1.27
    dollars = euros * facteur
    if(verbose): print euros,' euros = ',dollars,' dollars'
    return dollars

>>> conversion_euro_to_dollar(30)
30  euros =  38.1  dollars
38.1
>>> conversion_euro_to_dollar(30,verbose=False)
38.1
```

Fonction qui retourne plusieurs variables

En informatique une fonction peut retourner plusieurs variables à la fois par le mot clé **return**. Dans ce cas, la fonction renvoie une liste des variables de sortie.
Exemple 1 : Rotation d'un angle dans un espace euclidien à deux dimensions :

```
from math import cos, sin
def rotation(x,y,theta):
    u = x*cos(theta) + y*sin(theta)
    v = -x*sin(theta) + y*cos(theta)
    return u,v

>>> u,v = rotation(2,3,0.1) # la fonction renvoie 2 variables
>>> print u,v
2.2895085805 2.78534566254
>>> Aprime = rotation(2,3,0.1) # la fonction renvoie une liste
>>> print Aprime
(2.289508580496536, 2.785345662540 4207)
```

Graphiques

Une des forces du langage Python est la multiplicité de ses librairies disponibles gratuitement, et en particulier le module **matplotlib** qui permet de réaliser de jolis graphiques avec peu de lignes d'instruction.

Généralités sur matplotlib

Pour tracer un graphique simple, il suffit d'importer le module **pyplot** de matplotlib et d'utiliser la fonction **plot** :

```
from matplotlib import pyplot

x = [ ... ]
y = [ ... ]
fig = pyplot.figure()
pyplot.plot(x,y, options)
pyplot.show()
```

qui trace un graphe représentant les points d'abscisses contenus dans la liste x et d'ordonnées contenues dans la liste y. C'est l'opérateur **show()** qui déclenche l'apparition du graphique à l'écran.

Généralités sur matplotlib (suite)

Parmi les nombreuses options possibles pour la fonction plot, mentionnons celles-ci :

- **color** : fixe la couleur de la courbe, à choisir parmi 'green', 'blue', 'red', 'black',... ou 'r','g','b','k',...
- **linestyle** : fixe le style du trait, à choisir parmi '-' (trait continu), '–' (tireté), ' :' (pointillé) ou 'None' (aucun trait)
- **marker** : choisit le style des points parmi '.' (point), 'o' (rond), '*' (étoile), etc...
- **label** : donne un nom au tracé pour construire une légende par la suite

On peut **superposer** sans problème plusieurs graphiques en répétant l'instruction **plot**, et la construction de légende se fait automatiquement via **label**.

Généralités sur matplotlib (suite)

Plusieurs fonctions accompagnent pyplot pour enjoliver le graphique. Parmi les plus utiles, citons :

- **pyplot.xlim(xmin,xmax)** : limite l'axe des abscisses entre les valeurs xmin et xmax
- **pyplot.ylim(ymin,ymax)** : limite l'axe des ordonnées entre les valeurs ymin et ymax
- **pyplot.xscale('log')** : axe des abscisses en échelle logarithmique
- **pyplot.yscale('log')** : axe des ordonnées en échelle logarithmique
- **pyplot.xlabel("texte")** : donne un titre à l'axe des abscisses
- **pyplot.ylabel("texte")** : donne un titre à l'axe des ordonnées
- **pyplot.title("texte")** : donne un titre au graphique
- **pyplot.legend()** : trace une légende sur le graphique (utilise le mot clé label défini avec plot)

Tracé de fonction à une variable

Voici un exemple de tracé de la fonction parabole définie dans la section précédente :

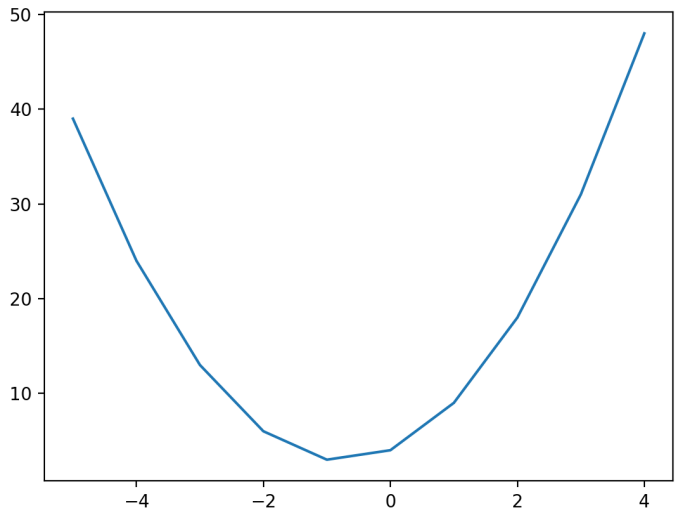
```
from matplotlib import pyplot

x_list = range(-5,5)
y_list = []

def parabole (x):
    f = 2*x**2+3*x+4
    return f

for x in x_list :
    y_list.append(parabole(x))

fig = pyplot.figure()
pyplot.plot(x_list,y_list)
pyplot.show()
```



Tracé de fonction à une variable (suite)

Deux remarques concernant ce premier graphique.

1. Tout d'abord il est pauvre en information, il faut lui rajouter des titres.
2. Ensuite, on remarque que la parabole n'est pas lisse pas constituée de segments de droite. Le module **pyplot** ne trace pas *stricto sensus* une fonction mathématique, mais un **nuage de points**, et ne se permet pas d'arrondir les courbes entre chaque point. Ceci est un problème de **discrétisation de la fonction** : celle-ci n'est représentée que par un ensemble discret de points.

Tracé de fonction à une variable (suite)

Pour améliorer le point 2, il faut un pas plus fin entre les points.

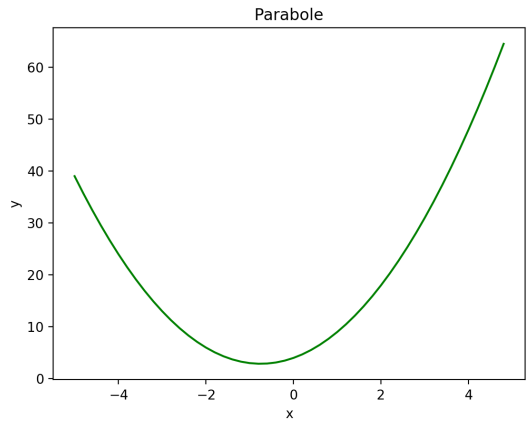
```
from matplotlib import pyplot
from numpy import linspace

xmin = -5.0
xmax = 5.0
npoints = 50
x_list=linspace(-5,5,50)
y_list=[]

def parabole (x):
    f = 2*x**2+3*x+4
    return f

for x in x_list:
    y_list.append(parabole(x))

fig = pyplot.figure()
pyplot.plot(x_list,y_list,color='g')
pyplot.title('Parabole')
pyplot.xlabel('x')
pyplot.ylabel('y')
pyplot.show()
```



Histogrammes

On trace des histogrammes très simplement avec la fonction **hist**. Celle-ci admet pour arguments principaux :

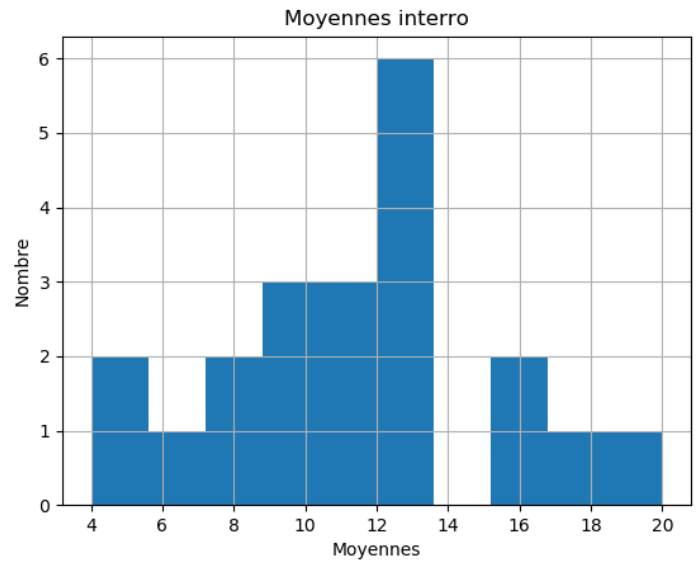
- la série de données dont on veut former l'histogramme
- les intervalles ou le nombre d'intervalle désiré par l'argument **bins**

Voici un exemple où on trace l'histogramme des moyennes à une interro avec 10 intervalles :

```
from matplotlib import pyplot

moyennes = [8.5, 12.5, 11, 10,
        13.5, 5, 18, 16.5, 12, 13,
        9, 10, 4, 20, 12.5, 10.5, 7
        8.5, 13.5, 15.5 ]

fig = pyplot.figure()
pyplot.hist(moyennes, bins=10)
pyplot.title('Moyennes interro')
pyplot.xlabel('Moyennes')
pyplot.ylabel('Nombre')
pyplot.grid(True)
pyplot.show()
```



Histogrammes (suite)

ou avec des intervalles de 5 points :

```
from matplotlib import pyplot

moyennes = [ 8.5, 12.5, 11, 10,
         13.5, 5, 18, 16.5, 12,
         13, 11, 9, 10, 4, 20,
         12.5, 10.5, 7, 8.5,
         13.5, 15.5 ]
bins = [0,5,10,15,20]
# intervalles de 5

fig = pyplot.figure()
pyplot.hist(moyennes, bins=bins)
pyplot.title('Moyennes interro')
pyplot.xlabel('Moyennes')
pyplot.ylabel('Nombre')
pyplot.grid(True)
pyplot.show()
```