Semestre 2, Lyon 1

Analyse II : Introduction à Python utile pour les maths

Rappel

1. Les variables

Les types primitifs en python sont les *int* (entier), *float* (réel), *bool* et les *string*. On peut savoir leurs types avec la fonction **type()** et les afficher avec **print()**

```
a = 42.0001

b = "Il est où le magneau ?"

c = 11

print(a, c)
print(type(b))
```

```
42.0001 11
<class 'str'>
```

Python est dynamique et faiblement typé, c'est-à-dire qu'on peut changer le type à n'importe quel moment et que les conversions de types sont possibles *int()*, *float()*, *round()*, *str()* etc.

```
message = "J'adore l'algèbre"
message = 2.6      # c'est mieux l'analyse
message = int(message)

print(message)
```

```
2
```

```
boolean = True
print(float(boolean))
```

```
1.0
```

Il est très pratique avec Python d'échanger les données de variables avec d'autres

```
a,b,c = 1,2,3
```

```
a,b,c = c,b,a
print(a,b,c)
```

```
3 2 1
```

2. Les opérations

Les 4 opérations arithmétiques sont présentes

```
x = 45
print(x + 2)
print(x - 2)
print(x * 3)

# des raccourcis comme tels sont possibles
x += 3    # équivalent x = x + 3
print(x)
x /= 2     # équivalent x = x / 2
print(x)

y = 2.5
print(x - y) # les opérateurs marchent entre float et int
print(y*10+x)
```

```
47
43
135
48
24.0
21.5
49.0
```

Remarquez tout de même qu'une opérateur entre int et float renvoie un float

L'exponenation, la puisssance, existe en python (et c'est rare !) avec **

```
3**3
```

```
27
```

Pour différencier la division décimale, Python nous offre l'opérateur // et %. // est la quotient de la division euclidienne (la partie enntière d'une /) % le modulo, le reste

```
print(15 %  7)
print(15 // 7)
```

```
1
2
```

La concaténation des strings peuvent se faire avec +

```
chaine = "John"
chaine += "athan"
chaine
```

```
{"type":"string"}
```

Il y a aussi les opérations booléennes, comme la négation, le ou et le e

```
True or False
```

```
True
```

```
False and True
```

```
False
```

```
p = True
not(p and False) or True
```

```
True
```

3. Les listes

Les listes est un ensembles d'éléments objets. On peut les concaténer, avoir accés aux ieme éléments, et pleins d'autres utilisations :

```
c = [1.23, 3, "Wech", True, 4]
print("Le deuxieme élément de c :", c[2])

d = [42]
c = c + d    # la concaténation
print(c)
```

```
Wech
[1.23, 3, 'Wech', True, 4, 42]
```

Pour ajouter un élément à la fin de la liste, on utilise la fonction **append()**

```
a = [42]

a.append(True)
a.append("I love you")

a
```

```
[42, True, 'I love you']
```

On calcule la longeur de la liste avec **len()**

```
print(len(a))
```

```
3
```

Python a un grand avantage (surtout en maths) **les slices** ! Les slices servent à afficher les sous tableaux de tableau de début à fin en faisant la tableau[debut:fin+1]

```
a = [10, 11, 12, 13, 14, 15, 16]
print(a[2:]) # affiche la sous liste à partir du 2ieme élément
print(a[:3]) # affiche de 0 à 2 ieme éléments (3ieme élément exclus)
print(a[1:4]) # affiche du 1 à 3 ieme éléments
```

```
[12, 13, 14, 15, 16]
[10, 11, 12]
[11, 12, 13]
```

```
print([0,1]*4) # fonctionnalité de multiplier les listes
```

```
[0, 1, 0, 1, 0, 1, 0, 1]
```

La fonction range

range(start,stop,step) retourne un objet de type range qui est une liste de nombre qui commence à start jusqu'a stop-1 avec un incrément step

```
print(list(range(0,5,1)))
```

```
[0, 1, 2, 3, 4]
```

Notez qu'on peut ommettre le start et ce dernier sera mit à 0 par défaut. De même pour step qui sera à 1

```
print(list(range(5)))
```

```
[0, 1, 2, 3, 4]
```

Exemples notables :

```
print(list(range(5,-1,-1)))
print(list(range(1,50,15)))
print(list(range(1,10)))
```

```
[5, 4, 3, 2, 1, 0]
[1, 16, 31, 46]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

4. if, elif, else

Le si, sinon si, sinon (la structure conditionnelle). ATTENTION : L'indétention est importante en python !

```
a = 42
if a%2 == 0 :
    # si a est divisible par deux alors ...
    print("C'est pair !")
elif a > 40:
    print("C'est impair mais au moins c'est plus grand que 40")
else:
    print("Rien à dire")
```

```
C'est pair !
```

A noter que elif et else sont optionnel à un if. Que if et elif attendent des booléens qui suivent. Les blocks sont reconnaissables à : suivis de l'indentation.

5. Boucle while

Rien à dire, c'est une boucle while suivis d'une condition, de : et d'une indentation. On répéte le block sous le while tant que la condition est vrai

```
continuer = True
i = 0
while continuer :
    i += 1
    if i > 40:
      continuer = False
print(i)
```

```
41
```

6. Boucle for

La boucle for en python s'écrit : for element in liste

Cette boucle va prendre élément par élément d'une liste et éxécutez ses instructions à l'intérieur d'elle

```
liste = [1,2,3,4,5]
for element in liste:
  print(element*element)
```

```
1
4
9
16
25
```

```
#  une boucle for plus classiques avec un rabge :
for element in range(10): # on peut dire i allant de 0 à 9 ici
  print(element+1)
```

```
1
2
3
4
5
6
7
8
9
10
```

Les listes par compréhension Après cela, en résulte les liste par compréhension. TRES PRATIQUE. Ne s'expliqu'avec des exemples :

```
[i for i in range(2,9,1)]
```

```
[2, 3, 4, 5, 6, 7, 8]
```

```
[i*i for i in range(11)]
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

7. Les fonctions

Les fonctions sont définies avec le mot clé "def" suivis du noms. Le code est précédé de : et d'une indétention

```
def f(x):
  k = 2
  return (x+k)**3 // 2

print(f(2))
```

```
32
```

Une fonction peut aussi ne rien retourner

```
def proc():
  for i in range(3):
    if i % 2 == 0:
      print("Je ne sais vraiment plus quoi écrire")
    else:
      print("Mais c'est pas grave :)")
```

Il peut y avoir des parametres qui sont par défeaut valué. Très utile pour vos fonctions. Ils doivent se trouver à la fin de la fonction.

```
def calcul_poid(m, g=0.91):
  return m*g

calcul_poid(40)
```

```
36.4
```

La récursivité est possible. Appeler la fonction à l'intérieur de la fonction

```
def foo_rec(x):
  if x<4:
    return 4
  elif x%2 == 0:
    return foo_rec(x/2) + 1
  else:
    return foo_rec(x-1)
```

Attention : les listes et les strings ne sont pas copiés quand on les passe dans une fonctions. Contrairement types primitifs. Regardez l'exemple ci-dessous on va changer le dernier élément d'une liste par effet de bord. (astuce : faire liste[-1] pour avoir le dernier élément)

```
ma_liste = [0,1,2,3,5]

def change_dernier(x, l):
  l[-1] = x

change_dernier(0, ma_liste)

print(ma_liste)
```

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

Quand on veut qu'il n'y ait pas d'effet de bord. La copie est nécessaire. Par exemple imaginons, on veut juste retourner la même liste mais avec les mêmes éléments au carré sans changer la liste originale, on ferait :

```
from copy import copy

l1 = [2,5,10]

def au_carre_liste(l):
  l2 = copy(l)
  return [e*e for e in l2]

print(au_carre_liste(l1))
print(l1)
```

```
[4, 25, 100]
[2, 5, 10]
```

Ainsi l1 n'a pas changé. On aurait pu aussi copier la liste l en faisant l2 = l[:]

Et pour les maths ?

1. NumPy: des matrice, toujours des matrices...

Le package **Numpy** est **LE package de référence** pour le calcul numérique en Python. Il doit être importé avec la commande suivante

```
import numpy as np
```

Fonctions usuelles

```
np.abs(-2) # la valeur absolue
np.sqrt(9) # racine carré
np.exp(2)
np.log(np.e)
np.sin(np.pi/6)
np.sin(np.pi/3)
```

```
0.8660254037844386
```

Ces fonctions s'appliquent aux tableaux qui s'appellent np.array

```
x = np.array([-2,23,-11,5])

y = np.abs(x)
print(y)
```

```
[ 2 23 11  5]
```

La fonction linspace(start,stop,n) permet de créer des tableaux qui va du start au stop en n nombre d'éléments

```
np.linspace(0,10,11)
```

```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

```
np.linspace(50,100,3)
```

```
array([ 50.,  75., 100.])
```

A ne pas confondre avec np.arange(start,stop,step) qui est équivalent à range(start,stop,step) !!

Les méthodes np.size(x) et np.shape(x) dont respectivevment le nombre d'élements de x et les dimensions de la matrice. Et il est possible de redimissioner avec np.reshape(x)

```
x = np.array([[1,3,5],[3,67,2]])

print("La matrice x : \n", x)
print(np.size(x))
print(np.shape(x))
print("Redimensionnage de la matrice x...")
x = x.reshape(6)
print(x)
```

```
La matrice x :
 [[ 1  3  5]
 [ 3 67  2]]
6
(2, 3)
Redimensionnage de la matrice x...
[ 1  3  5  3 67  2]
```

Regardez attentivement ces exemples de tableaux

```
c = np.array([1, -2, 7, 0, 10])
2 * c
print(2 * c)
c.size
np.size(c)
c[3]
c[0] + c[1] + c[2] + c[3] + c[4]
c[0] = 0
print(c)
d = np.array([[0, 0, 0], [0, 0, 0]])
print(d)
d = np.zeros((2, 3))
print(d)
d.shape
np.shape(d)
a = np.array([[0, 1], [0, 0]])
print(a)
a[1, 1] = 1
a[0, 0] = a[1, 1]
print(a)
```

```
[ 2 -4 14  0 20]
[ 0 -2  7  0 10]
[[0 0 0]
 [0 0 0]]
[[0. 0. 0.]
 [0. 0. 0.]]
[[0 1]
 [0 0]]
```

```
[[1 1]
 [0 1]]
```

Création automatique de tableaux structurés

```
print(np.arange(0,10))
print(np.arange(0,10,0.5))
print(np.linspace(0, 10, 10))
print(np.linspace(0, 10, 11))
print(np.linspace(0, 10, 21))
print(np.linspace(0,2*np.pi,7))
x = np.array([2,-1,np.pi,-3,-2,-np.e])
print(x)
print(np.max(x))
print(np.min(x))
x = np.arange(0,20)
print(x)
print(x.size)
```

```
[0 1 2 3 4 5 6 7 8 9]
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.
8.5
 9.  9.5]
[ 0.          1.11111111  2.22222222  3.33333333  4.44444444
5.55555556
  6.66666667  7.77777778  8.88888889 10.        ]
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[ 0.   0.5  1.   1.5  2.   2.5  3.   3.5  4.   4.5  5.   5.5  6.   6.5
  7.   7.5  8.   8.5  9.   9.5 10. ]
[0.         1.04719755 2.0943951  3.14159265 4.1887902  5.23598776
 6.28318531]
[ 2.         -1.          3.14159265 -3.         -2.         -
2.71828183]
3.141592653589793
-3.0
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
20
```

Les slices sont possibles aussi sur les Arrays. (cf les slices sur le chapitre des litstes)

```
print(x[2:17])
print(x[2])
print(x[17])
print(x[:-2])
print(x[3:])
print(x[9:-8])
```

```
[ 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
2
```

```
17
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17]
[ 3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
[ 9 10 11]
```

2. MathPlotLib : Tracer des graphiques

Il existe beaucoup de packages en Python pour tracer des graphiques. Le plus utilisé est matplotlib. Pour charger les fonctions principales de ce package, on utilise la commande suivante :

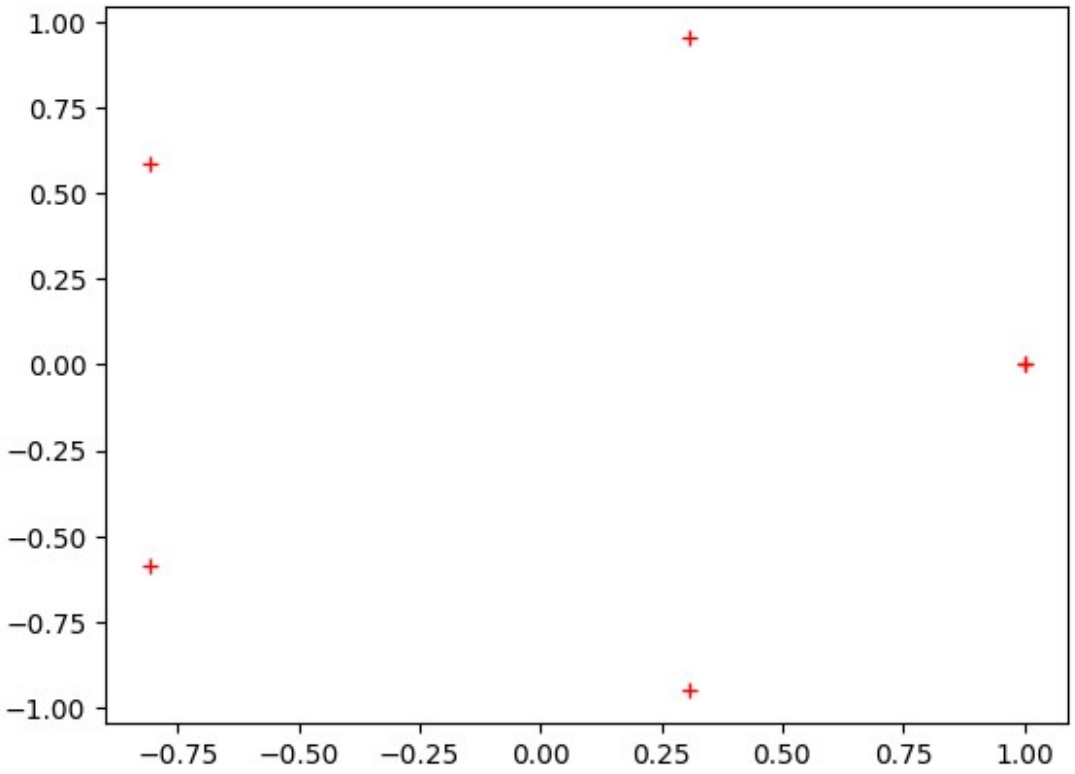
```
import matplotlib.pyplot as plt
```

Nuage de points et points reliés

Pour représenter avec Python un nuage de points {(xk, yk): 1 ≤ k ≤ N}, on définit un vecteur contenant les abscisses (x1, . . . , xN) et un vecteur contenant les ordonnées (y1, . . . , yN) des points dans le même ordre. On ajoute ensuite le marqueur souhaité pour matérialiser les points (une croix +) dans la couleur souhaitée (r pour 'red').

```
x = [np.cos(2*i*np.pi/5) for i in range(6)]
y = [np.sin(2*i*np.pi/5) for i in range(6)]
plt.plot(x, y, 'r+')
```

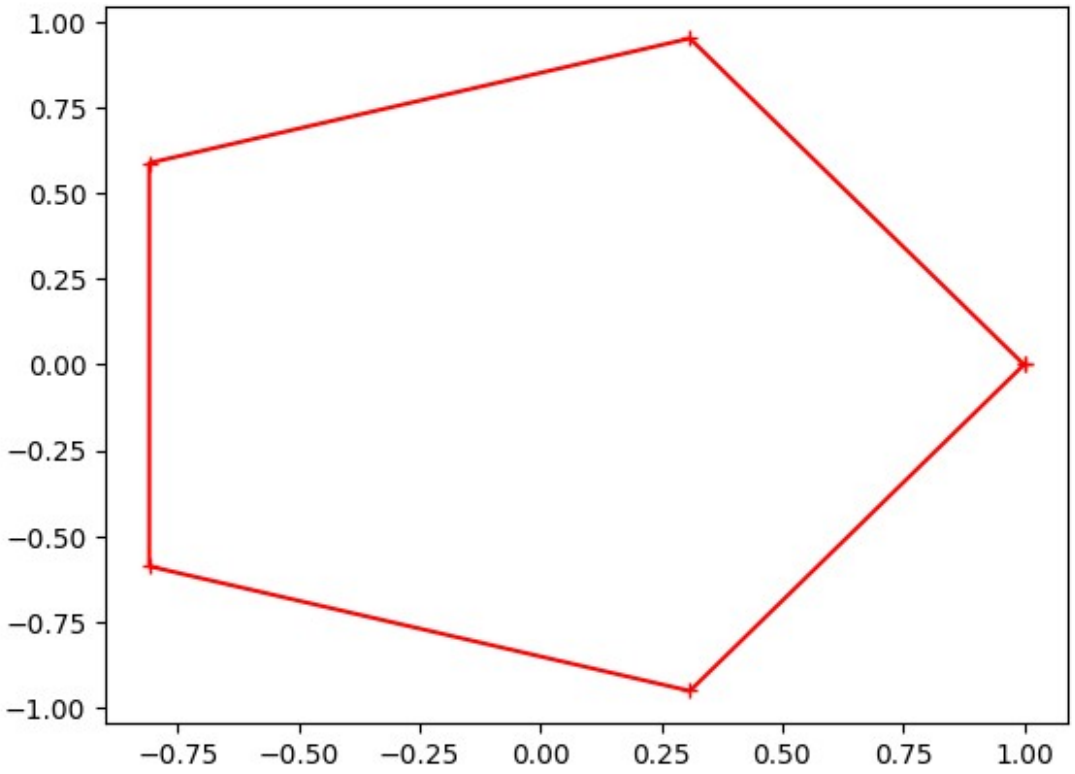
```
[<matplotlib.lines.Line2D at 0x79a56d8902b0>]
```



Pour relier les points, il suffit d'ajouter le type de trait par lequel on souhaite que les points soient reliés (- pour un trait continu).

```
x = [np.cos(2*i*np.pi/5) for i in range(6)]
y = [np.sin(2*i*np.pi/5) for i in range(6)]
plt.plot(x, y, 'r+-')
```

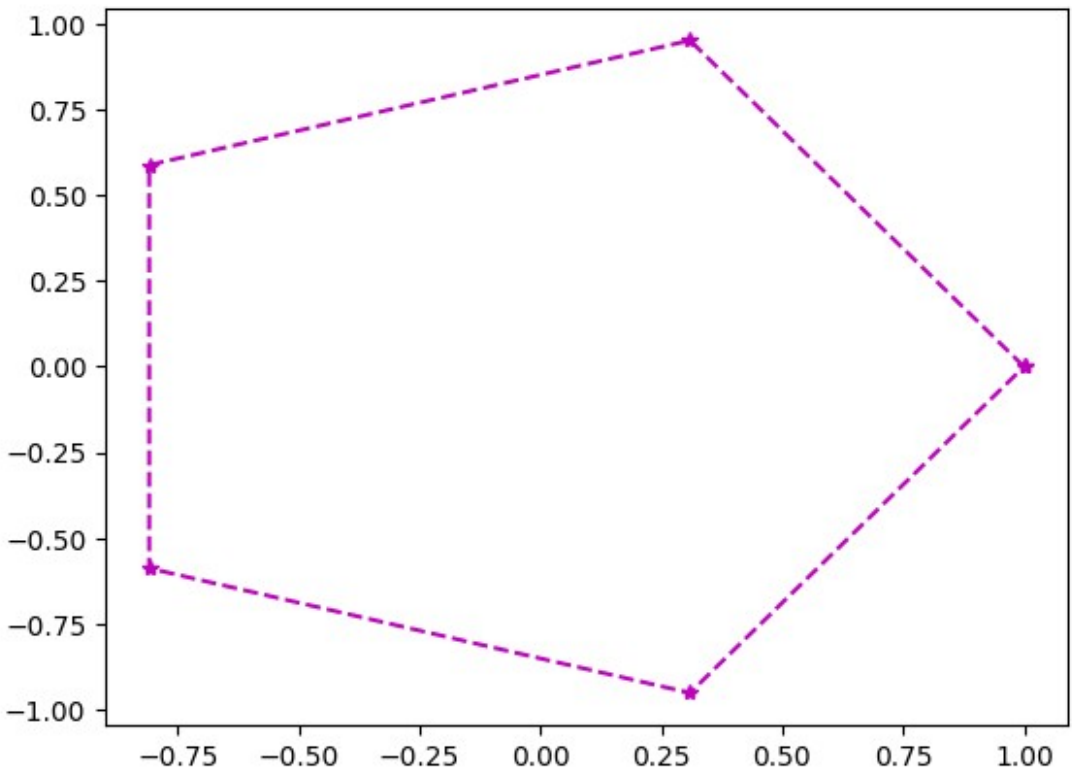
```
[<matplotlib.lines.Line2D at 0x79a56d4a1ae0>]
```



D'autres options permettent de spécifier la couleur de la courbe (m magenta), la représentation des coordonnées (* étoile) et le type du trait (-- discontinu).

```
x = [np.cos(2*i*np.pi/5) for i in range(6)]
y = [np.sin(2*i*np.pi/5) for i in range(6)]
plt.plot(x, y, 'm*--')
```

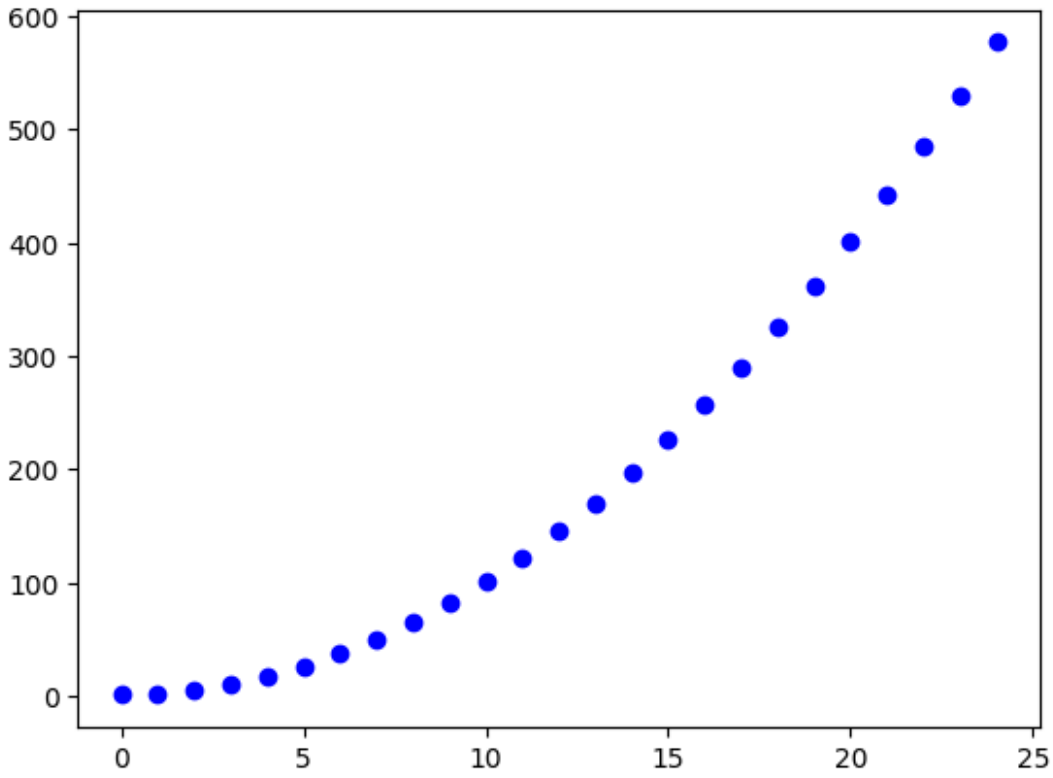
```
[<matplotlib.lines.Line2D at 0x79a56b369a50>]
```



Représentation des termes d'une suite Pour visualiser les N premiers termes d'une suite (un)n∈N, on peut dessiner le nuage de points {(n, un): 0 ≤ n ≤ N} (réliés ou non). Pour cela, on commence par définir le vecteur des abscisses (0, . . . , N) puis le vecteur des ordonnées (u0, . . . , uN). On peut finalement tracer la suite en précisant que les marqueurs sont par exemple des ronds (o) bleus (b). Dans l'exemple qui suit, la suite (un)n∈N est définie par un = n^2 + 1 pour tout n ∈ N.

```
n = np.arange(25)

def u(n):
  return n**2+1

plt.plot(n,u(n), 'bo')
```

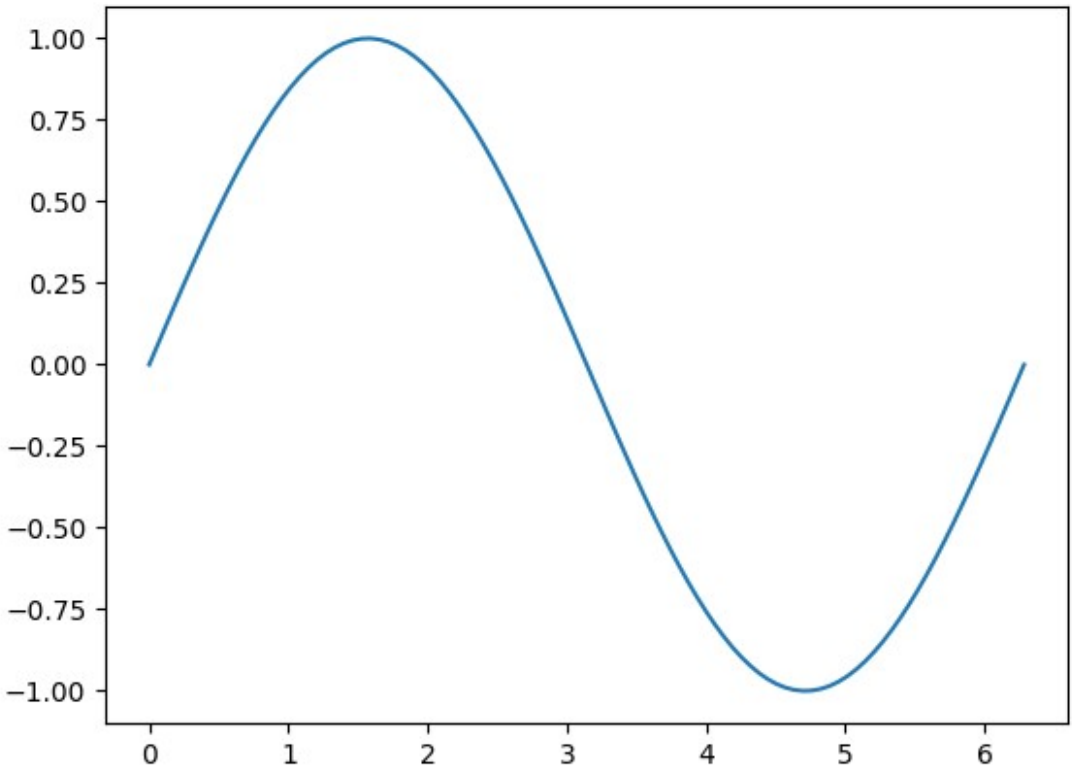
```
[<matplotlib.lines.Line2D at 0x79a56b3fd390>]
```



Tracé de la courbe représentative d'une fonction Pour tracer le graphe d'une fonction f : Df → R, on choisit d'abord un segment [a, b] inclus dans Df , où vont être prises les abscisses. On ne peut pas demander à Python de calculer une infinité de valeurs, on va donc créer un vecteur d'abscisses contenant "beaucoup" de points dans [a, b], (x0, . . . , xN) avec a = x0 < x1 < . . . < xN = b (par exemple une centaine). Ensuite, on va créer le vecteur des ordonnées correspondantes (f(x0), . . . , f(xN)). Enfin, on va tracer les points reliés {(xk, f(xk)): 0 ≤ k ≤ N}. Dans ce cas, on ne souhaite pas voir les points matérialisés, on indique donc juste le fait que l'on souhaite un trait continu reliant les points (xk, f(xk)).

```
x = np.linspace(0, 2*np.pi, 100)
plt.plot(x, np.sin(x))
```

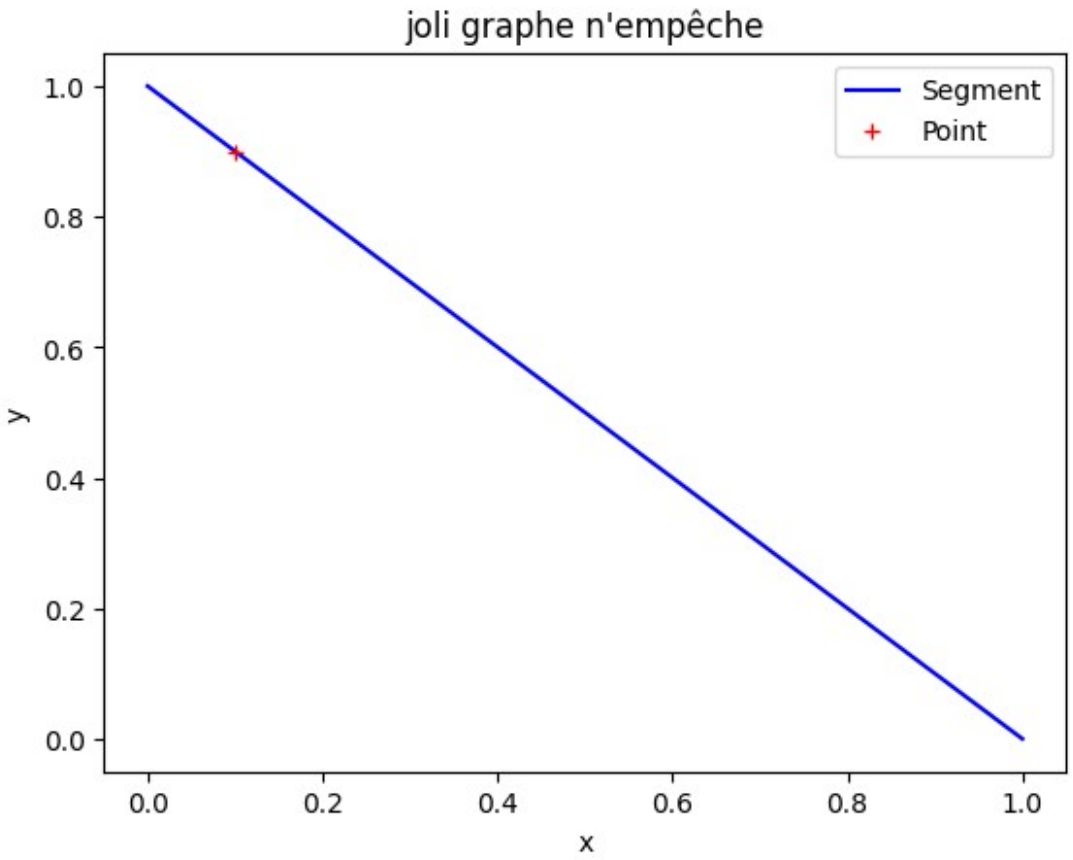
```
[<matplotlib.lines.Line2D at 0x79a56b182020>]
```



Titres et légendes Toute figure doit être accompagnée d'une légende, et d'un titre pour chacun des axes et pour la figure.

```
plt.plot([0, 1], [1, 0], 'b', label='Segment')
plt.plot(0.1, 0.9,'r+', label='Point')
plt.xlabel('x')
plt.ylabel('y')
plt.title("joli graphe n'empêche")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x79a56b2bab00>
```



Exercice

Le but de l'exercice est de rapidement calculer une intégrale d'une fonction positive avec la méthode de Riemmann des trapèzes

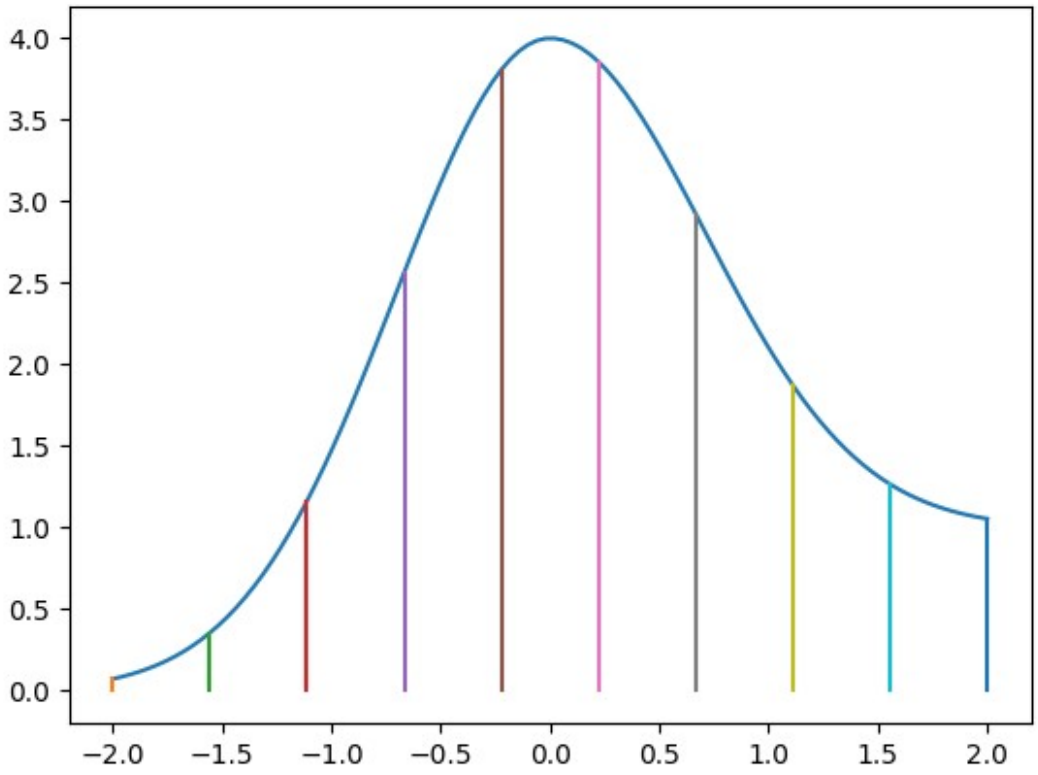
Question 1 : Définir la fonction gaussienne f(x) = {4exp(-x^2) si x < 0, 3exp(-x^2) + 1 sinon} en python. Afficher quelques images.

```
# Solution
def gauss(x):
  if x < 0:
    return 4*np.exp(-(x*x))
  else:
    return 3*np.exp(-(x*x))+1

print(gauss(-4),gauss(5.2))
```

```
4.5014069887703647e-07 1.0000000000054174
```

Question 2 : Afficher cette fonction sur -2 à 2. Par suite, afficher 10 trapèzes qui suivent la courbe de la fonciton

```
x = np.linspace(-2,2,100)
y = [gauss(i) for i in x]
plt.plot(x,y)

x = np.linspace(-2,2,10)
for i in x:
  plt.plot([i,i],[0,gauss(i)])
```



Question 3 : Faire un programme qui calcule la somme des aires de ces trapèzes (qui correspond à une approximation de l'intégrale de Riemmann de la fonction)

Rappel : Aire du trapèez : (B + b)*H/2 avec B grande base, b la petite base et H la hauteur

```
x = np.linspace(-2,2,10)

H = np.abs(x[1]-x[0])

aire = 0
i = -2
```

```
while i < 2:
  aire += (gauss(i)+gauss(i+H))*H/2
  i += H

print(aire)
```

```
8.624754544902922
```

Question Bonus

Enigme : Edouard a le triple de l'âge de Marie. Philippe a 15 ans de plus que Edouard. Zina a un an de plus que Philippe. Si Zina a un age paire alors elle a le même que Marie, sinon un an de plus. Faire une fonction qui prend en paramêtre l'âge de Zina et retourne l'âge de Marie.