018 - Les bibliothèques et les modules

September 3, 2023

1 Importer une bibliothèque

1.1 Importer une bibliothèque sans Alias

[ ]:
```
# La bibliothèque math contient des fonctions et des constantes mathématiques
import math
print(math.sin(0))
print(math.cos(0))
print(math.pi)
```

```
0.0
1.0
3.141592653589793
```

1.2 Importer une bibliothèque avec un Alias

[ ]:
```
import math as m
print(m.sin(0))
print(m.cos(0))
print(m.pi)
```

```
0.0
1.0
3.141592653589793
```

2 Importer une fonction ou une variable d'une bibliothèque

2.1 Importer une seule variable/fonction

[ ]:
```
from math import cos
print(cos(0))
```

```
1.0
```

2.2 Importer plusieurs variables/fonctions

[ ]:
```
from math import cos,sin,pi
print(cos(pi))
print(sin(pi))
```

```
-1.0
1.2246467991473532e-16
```

2.3 Importer toutes les fonctions et les variables d'une bibliothèque

[ ]:
```
from math import *
print(cos(pi))
print(log(e))
```

```
-1.0
1.0
```

3 Différence entre importer une fonction et importer une bibliothèque

[ ]:
```
# On importe la bibliothèque math
import math as m
e=20
print(m.log(m.e))
```

```
1.0
```

[ ]:
```
# On importe la fonction log et la variable e de la bibliothèque math
from math import log,e
e=20
print(log(e))
```

```
2.995732273553991
```

4 La bibliothèque math

##Les fonctions à retenir de la bibliothèque math - sin(x), cos(x), et tan(x) (Le x doit être en radian). - asin, acos, atan (Les fonctions réciproques de sin, cos et tan). - exp(x) pour exponentiel. - log(x,b) : calcul le log à base b de x. - log(x) : calcul ln(x). - sinh(x), cosh(x) et tanh(x). - asinh(x), acosh(x) et atanh(x) (Les fonctions réciproques de sinh, cosh et tanh). - sqrt(x), c'est la racine carrée de x.

##Les variable à retenir de la bibliothèque math - e : constante de Néper ou Nombre d'euler c'est exp(1) ; - pi : constante d'archimède : 3,141592653589793

5 La bibliothèque numpy

NumPy est une bibliothèque pour langage de programmation Python, destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux.

[ ]:
```
import numpy as np
```

[ ]:
```
# Création d'un array unidimensionnel
V = np.array([12,2,5,9])
print(V)
```

```
[12  2  5  9]
```

[ ]:
```
# Création d'un array multidimensionnel (Matrice)
M = np.array([[12,2,5,9],[2,9,0,4],[7,0,1,3]])
print(M)
```

```
[[12  2  5  9]
 [ 2  9  0  4]
 [ 7  0  1  3]]
```

[ ]:
```
# Accéder aux éléments d'un vecteur
V = np.array([12,2,5,9,18,9,0])
print("V[0] =",V[0])
print("V[-1] =",V[-1])
print("V[4] =",V[4])
```

```
V[0] = 12
V[-1] = 0
V[4] = 18
```

[ ]:
```
# Accéder aux éléments d'une matrice
M = np.array([[1,4,7,9,0],[45,2,9,0,12],[12,9,5,3,2]])
print(M)
print()
print("M[0,1] =",M[0,1])
print("M[2,4] =",M[2,4])
print("M[1,3] =",M[1,3])
```

```
[[ 1  4  7  9  0]
 [45  2  9  0 12]
 [12  9  5  3  2]]

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

[ ]:
```
# Extraction des sous vecteurs
V=np.array([4,5,7,8,4,5])
```

```
print("V[1:5] =",V[1:5])
print("V[:3] =",V[:3])
print("V[2:] =",V[2:])
print("V[:-1] =",V[:-1])
```

```
V[1:5] = [5 7 8 4]
V[:3] = [4 5 7]
V[2:] = [7 8 4 5]
V[:-1] = [4 5 7 8 4]
```

[ ]:
```
# Extraction des sous matrices
M = np.
  ↪array([[1,4,7,9,0,8],[45,2,9,0,12,9],[12,9,5,3,2,10],[1,4,7,9,0,1],[12,9,5,3,2,10]])
print(M)
print()
print("M[:3,:3] =",M[:3,:3],sep='\n')
print("M[1:4,1:5] =",M[1:4,1:5],sep='\n')
```

```
[[ 1  4  7  9  0  8]
 [45  2  9  0 12  9]
 [12  9  5  3  2 10]
 [ 1  4  7  9  0  1]
 [12  9  5  3  2 10]]

M[:3,:3] =
[[ 1  4  7]
 [45  2  9]
 [12  9  5]]
M[1:4,1:5] =
[[ 2  9  0 12]
 [ 9  5  3  2]
 [ 4  7  9  0]]
```

[ ]:
```
# Extraction des lignes et des colonnes
M = np.
  ↪array([[1,4,7,9,0,8],[45,2,9,0,12,9],[12,9,5,3,2,10],[1,4,7,9,0,1],[12,9,5,3,2,10]])
print(M)
print()
print("La deuxième ligne est :",M[1])
print("La troisième colonne est :",M[:,2])
```

```
[[ 1  4  7  9  0  8]
 [45  2  9  0 12  9]
 [12  9  5  3  2 10]
 [ 1  4  7  9  0  1]
 [12  9  5  3  2 10]]

La deuxième ligne est : [45  2  9  0 12  9]
La troisième colonne est : [7 9 5 7 5]
```

[ ]:
```
# Opérations entre array et scalaire (int ou float)
M = np.array([[1,4,7,9,0],[45,2,9,0,12],[12,9,5,3,2]])
print(M)
print()
print(2*M)
```

```
[[ 1  4  7  9  0]
 [45  2  9  0 12]
 [12  9  5  3  2]]

[[ 2  8 14 18  0]
 [90  4 18  0 24]
 [24 18 10  6  4]]
```

[ ]:
```
# Opérations entre array et scalaire (int ou float)
M = np.array([[1,4,7,9,0],[45,2,9,0,12],[12,9,5,3,2]])
print(M)
print()
print(1.5*M)
```

```
[[ 1  4  7  9  0]
 [45  2  9  0 12]
 [12  9  5  3  2]]

[[ 1.5  6.  10.5 13.5  0. ]
 [67.5  3.  13.5  0.  18. ]
 [18.  13.5  7.5  4.5  3. ]]
```

[ ]:
```
# Opérations entre array et scalaire (int ou float)
M = np.array([[1,4,7,9,0],[45,2,9,0,12],[12,9,5,3,2]])
print(M)
print()
print(M**2)
```

```
[[ 1  4  7  9  0]
 [45  2  9  0 12]
 [12  9  5  3  2]]

[[   1   16   49   81    0]
 [2025    4   81    0  144]
 [ 144   81   25    9    4]]
```

[ ]:
```
# Opérations entre array et scalaire (int ou float)
M = np.array([[1,4,7,9,0],[45,2,9,0,12],[12,9,5,3,2]])
print(M)
print()
print(M+10)
```

```
[[ 1  4  7  9  0]
```

```
 [45  2  9  0 12]
 [12  9  5  3  2]]

[[11 14 17 19 10]
 [55 12 19 10 22]
 [22 19 15 13 12]]
```

[ ]:
```
# Opérations entre array et scalaire (int ou float)
M = np.array([[1,4,7,9,0],[45,2,9,0,12],[12,9,5,3,2]])
print(M)
print()
print(M-5)
```

```
[[ 1  4  7  9  0]
 [45  2  9  0 12]
 [12  9  5  3  2]]

[[-4 -1  2  4 -5]
 [40 -3  4 -5  7]
 [ 7  4  0 -2 -3]]
```

[ ]:
```
# Opérations entre array et scalaire (int ou float)
M = np.array([[1,4,7,9,0],[45,2,9,0,12],[12,9,5,3,2]])
print(M)
print()
print(M/5)
```

```
[[ 1  4  7  9  0]
 [45  2  9  0 12]
 [12  9  5  3  2]]

[[0.2 0.8 1.4 1.8 0. ]
 [9.  0.4 1.8 0.  2.4]
 [2.4 1.8 1.  0.6 0.4]]
```

[ ]:
```
# Opérations entre array et scalaire (int ou float)
M = np.array([[1,4,7,9,0],[45,2,9,0,12],[12,9,5,3,2]])
print(M)
print()
print(M//5)
```

```
[[ 1  4  7  9  0]
 [45  2  9  0 12]
 [12  9  5  3  2]]

[[0 0 1 1 0]
 [9 0 1 0 2]
 [2 1 1 0 0]]
```

[ ]:
```
# Opérations entre array et scalaire (int ou float)
M = np.array([[1,4,7,9,0],[45,2,9,0,12],[12,9,5,3,2]])
print(M)
print()
print(M%5)
```

```
[[ 1  4  7  9  0]
 [45  2  9  0 12]
 [12  9  5  3  2]]

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

[ ]:
```
# Comparaison d'un array avec un scalaire (int ou float)
M = np.array([[1,4,7,9,0],[45,2,9,0,12],[12,9,5,3,2]])
print(M)
print()
print(M>6)
```

```
[[ 1  4  7  9  0]
 [45  2  9  0 12]
 [12  9  5  3  2]]

[[False False  True  True False]
 [ True False  True False  True]
 [ True  True False False False]]
```

[ ]:
```
# Opérations entre deux arrays de meme dimensions
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
M2=np.array([[0,5,2],[2,8,9],[1,8,2]])
print(M1+M2)
```

```
[[ 1  7  5]
 [ 6 13 15]
 [ 8 16 11]]
```

[ ]:
```
# Opérations entre deux arrays de meme dimensions
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
M2=np.array([[0,5,2],[2,8,9],[1,8,2]])
print(M1-M2)
```

```
[[ 1 -3  1]
 [ 2 -3 -3]
 [ 6  0  7]]
```

[ ]:
```
# Opérations entre deux arrays de meme dimensions
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
M2=np.array([[0,5,2],[2,8,9],[1,8,2]])
```

```
print(M1*M2)
```

```
[[ 0 10  6]
 [ 8 40 54]
 [ 7 64 18]]
```

[ ]:
```
# Opérations entre deux arrays de meme dimensions
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
M2=np.array([[0,5,2],[2,8,9],[1,8,2]])
print(M1**M2)
```

```
[[       1       32        9]
 [      16   390625 10077696]
 [       7 16777216       81]]
```

[ ]:
```
# Opérations entre deux arrays de meme dimensions
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
M2=np.array([[0,5,2],[2,8,9],[1,8,2]])
print(M2/M1)
```

```
[[0.         2.5        0.66666667]
 [0.5        1.6        1.5       ]
 [0.14285714 1.         0.22222222]]
```

[ ]:
```
# Opérations entre deux arrays de meme dimensions
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
M2=np.array([[0,5,2],[2,8,9],[1,8,2]])
print(M2//M1)
```

```
[[0 2 0]
 [0 1 1]
 [0 1 0]]
```

[ ]:
```
# Opérations entre deux arrays de meme dimensions
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
M2=np.array([[0,5,2],[2,8,9],[1,8,2]])
print(M2%M1)
```

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

Toute les fonctions et les variables vus dans la bibliothèque math sont disponible sur la bibliothèque numpy. Les fonctions mathématiques définies sur numpy sont applicables meme aux arrays unidimensionnels (Vecteurs) et multidimensionnels (Matrices)

[ ]:
```
# Appliquer une fonction mathématique sur un array
M2=np.array([[0,5,2],[2,np.pi,9],[1,8,2]])
print(np.cos(M2))
```

```
[[ 1.          0.28366219 -0.41614684]
 [-0.41614684 -1.         -0.91113026]
 [ 0.54030231 -0.14550003 -0.41614684]]
```

[ ]:
```
# Appliquer une fonction mathématique sur un array
def f(x):
  return x**2-5*x+4

M2=np.array([[0,5,2],[2,0,9],[1,8,2]])
print(f(M2))
```

```
[[ 4  4 -2]
 [-2  4 40]
 [ 0 28 -2]]
```

[ ]:
```
# Taille d'une matrice
M2=np.array([[0,5,2],[2,0,9],[1,8,2]])
M2.size
```

[ ]:
```
9
```

[ ]:
```
# Taille d'une matrice
M2=np.array([[0,5,2],[2,0,9],[1,8,2]])
np.size(M2)
```

[ ]:
```
9
```

[ ]:
```
# Forme d'une matrice
M2=np.array([[0,5],[2,0],[1,8]])
M2.shape
```

[ ]:
```
(3, 2)
```

[ ]:
```
# Forme d'une matrice
M2=np.array([[0,5],[2,0],[1,8]])
np.shape(M2)
```

[ ]:
```
(3, 2)
```

[ ]:
```
# La fonction linspace
L = np.linspace(1,2,101)
print(L)
```

```
[1.   1.01 1.02 1.03 1.04 1.05 1.06 1.07 1.08 1.09 1.1  1.11 1.12 1.13
 1.14 1.15 1.16 1.17 1.18 1.19 1.2  1.21 1.22 1.23 1.24 1.25 1.26 1.27
 1.28 1.29 1.3  1.31 1.32 1.33 1.34 1.35 1.36 1.37 1.38 1.39 1.4  1.41
 1.42 1.43 1.44 1.45 1.46 1.47 1.48 1.49 1.5  1.51 1.52 1.53 1.54 1.55
 1.56 1.57 1.58 1.59 1.6  1.61 1.62 1.63 1.64 1.65 1.66 1.67 1.68 1.69
 1.7  1.71 1.72 1.73 1.74 1.75 1.76 1.77 1.78 1.79 1.8  1.81 1.82 1.83
```

```
 1.84 1.85 1.86 1.87 1.88 1.89 1.9  1.91 1.92 1.93 1.94 1.95 1.96 1.97
 1.98 1.99 2.  ]
```

[ ]:
```
# La fonction arange
P = np.arange(1,2,0.1)
print(P)
```

```
[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9]
```

[ ]:
```
# La fonction arange
P = np.arange(1,2.001,0.1)
print(P)
```

```
[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ]
```

[ ]:
```
# Déclarer un vecteur de zeros
Z = np.zeros(6)
print(Z)
```

```
[0. 0. 0. 0. 0. 0.]
```

[ ]:
```
# Déclarer une matrice de zeros
ZM = np.zeros((5,3))
print(ZM)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

[ ]:
```
# Déclarer un vecteur de uns
O = np.ones(6)
print(O)
```

```
[1. 1. 1. 1. 1. 1.]
```

[ ]:
```
# Déclarer une matrice de uns
OM = np.ones((5,3))
print(OM)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

[ ]:
```
# Créer la matrice carrée d'identité
I = np.eye(7)
print(I)
```

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

[ ]:
```
# La fonction transpose()
M = np.array([[1,4,7,9,0],[45,2,9,0,12],[12,9,5,3,2]])
print(M)
print()
print(np.transpose(M))
```

```
[[ 1  4  7  9  0]
 [45  2  9  0 12]
 [12  9  5  3  2]]

[[ 1 45 12]
 [ 4  2  9]
 [ 7  9  5]
 [ 9  0  3]
 [ 0 12  2]]
```

[ ]:
```
# Le produit scalaire des vecteur à l'aide de la fonction vdot
# ATTENTION, LES VECTEURS DOIVENT ETRE DE MEME TAILLE
V1 = np.array([3,8,2,1])
V2 = np.array([2,3,1,1])
np.vdot(V1,V2)
```

[ ]:
```
33
```

[ ]:
```
# Le produit matriciel à l'aide de la fonction dot
# ATTENTION, LE PRODUIT MATRICIEL N'EST PAS COMMUTATIF
# ATTENTION, LE NOMBRE DE COLONNES DE LA PREMIERE MATRICE DOIT ETRE EGAL AU␣
 ↪NOMBRE DE LIGNES DE LA DEUXIEME MATRICE
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
M2=np.array([[0,5,2],[2,8,9],[1,8,2]])
np.dot(M1,M2)
```

[ ]:
```
array([[  7,  45,  26],
       [ 16, 108,  65],
       [ 25, 171, 104]])
```

[ ]:
```
# Le produit matriciel à l'aide de la fonction dot
# ATTENTION, LE PRODUIT MATRICIEL N'EST PAS COMMUTATIF
# ATTENTION, LE NOMBRE DE COLONNES DE LA PREMIERE MATRICE DOIT ETRE EGAL AU␣
 ↪NOMBRE DE LIGNES DE LA DEUXIEME MATRICE
```

```
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
M2=np.array([[0,5,2],[2,8,9],[1,8,2]])
np.dot(M2,M1)
```

[ ]:
```
array([[ 34,  41,  48],
       [ 97, 116, 135],
       [ 47,  58,  69]])
```

[ ]:
```
# Le max des elements d'un array
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
np.max(M1)
```

[ ]:
```
9
```

[ ]:
```
# Le min des elements d'un array
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
np.min(M1)
```

[ ]:
```
1
```

[ ]:
```
# La moyenne des elements d'un array
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
np.mean(M1)
```

[ ]:
```
5.0
```

[ ]:
```
# La somme des elements d'un array
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
np.sum(M1)
```

[ ]:
```
45
```

[ ]:
```
# La somme des elements d'un array multidimensionnel par colonne
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(sum(M1))
```

```
[12 15 18]
```

[ ]:
```
# Le determinant d'une matrice
M1=np.array([[1,2,3],[4,5,6],[7,8,9]])
M2=np.array([[0,5,2],[2,8,9],[1,8,2]])
print("Determinant de M1 :",np.linalg.det(M1))
print("Determinant de M2 :",np.linalg.det(M2))
```

```
Determinant de M1 : 0.0
Determinant de M2 : 40.999999999999986
```

[ ]:
```
# L'inverse d'une matrice
print("Le determinant de M2 est non nul")
```

```
print("L'inverse de M2 est :")
print(np.linalg.inv(M2))
```

```
Le determinant de M2 est non nul
L'inverse de M2 est :
[[-1.36585366  0.14634146  0.70731707]
 [ 0.12195122 -0.04878049  0.09756098]
 [ 0.19512195  0.12195122 -0.24390244]]
```

[ ]:
```
# Résolution d'un système linéaire AX=B
A=np.array([[0,5,2],[2,8,9],[1,8,2]])
B=np.array([3,8,5])
print("La solution de AX=B est :")
print(np.linalg.solve(A,B))
```

```
La solution de AX=B est :
[0.6097561  0.46341463 0.34146341]
```

[ ]:
```
# Racine du polynome x**5+3*x**2-x+4
print(np.roots([1,0,0,3,-1,4]))
```

```
[-1.70556266+0.j          0.91845525+1.04877358j  0.91845525-1.04877358j
 -0.06567392+1.09654871j -0.06567392-1.09654871j]
```

6 La bibliothèque scipy

6.1 La bibliothèque scipy.optimize

Cette bibliothèque permet la résolution des equation f(x)=0

[ ]:
```
import scipy.optimize as so
```

[ ]:
```
# Supposons vouloir résoudre l'équation x²-4x+1 = 0 dans l'intervalle [0,1]
def f(x):
  return x**2-4*x+1

s=so.bisect(f,0,1)
print("La solution de l'équation f(x) = 0 à l'aide de la fonction bisect est :
 ↪",s)
print("f(",s,") =",f(s))

# La fonction bisect utilise la méthode de la dichotomie
```

```
La solution de l'équation f(x) = 0 à l'aide de la fonction bisect est :
0.2679491924300237
f( 0.2679491924300237 ) = 3.807176796044587e-12
```

[ ]:
```
# La fonction newton nécéssite la dérivée
def df(x):
```

```
  return 2*x-4

sn=so.newton(f,0,df)
print("La solution de l'équation f(x) = 0 à l'aide de la fonction newton est :
 ↪",sn)
print("f(",sn,") =",f(sn))

# La fonction newton utilise la méthode de  Newton si on lui fournit la dérivée
```

```
La solution de l'équation f(x) = 0 à l'aide de la fonction newton est :
0.2679491924311227
f( 0.2679491924311227 ) = 0.0
```

[ ]:
```
# La fonction newton sans dérivée
sl=so.newton(f,0)
print("La solution de l'équation f(x) = 0 à l'aide de la fonction newton est :
 ↪",sl)
print("f(",sl,") =",f(sl))
```

```
La solution de l'équation f(x) = 0 à l'aide de la fonction newton est :
0.2679491924311181
f( 0.2679491924311181 ) = 1.5987211554602254e-14
```

6.2 La bibliothèque scipy.integrate

La bibliothèque scipy.integrate permet le calcul des integrales et la résolutions des equations différentielles

[ ]:
```
import scipy.integrate as si
```

[ ]:
```
# Calculons l'intégrale de la fonction g(x)=5x+1 de 0 à 1
def g(x):
  return 5*x+1

i = si.quad(g,0,1)
print("L'integral de g de 0 à 1 est",i[0])
```

```
L'integral de g de 0 à 1 est 3.5
```

[ ]:
```
# Nous voulons résoudre l'équation différentielle 2y'(t) - y(t) - t = 0 sur␣
 ↪l'intervalle [1,2] avec y(1)=5
# L'equa diff est équivalente à y'(t) = (y(t) + t)/2 = F(y(t),t)
# Avec F(p,q) = (p+q)/2
# On définit F
def F(p,q):
  return (p+q)/2

# On crée la liste des abscisses
import numpy as np
```

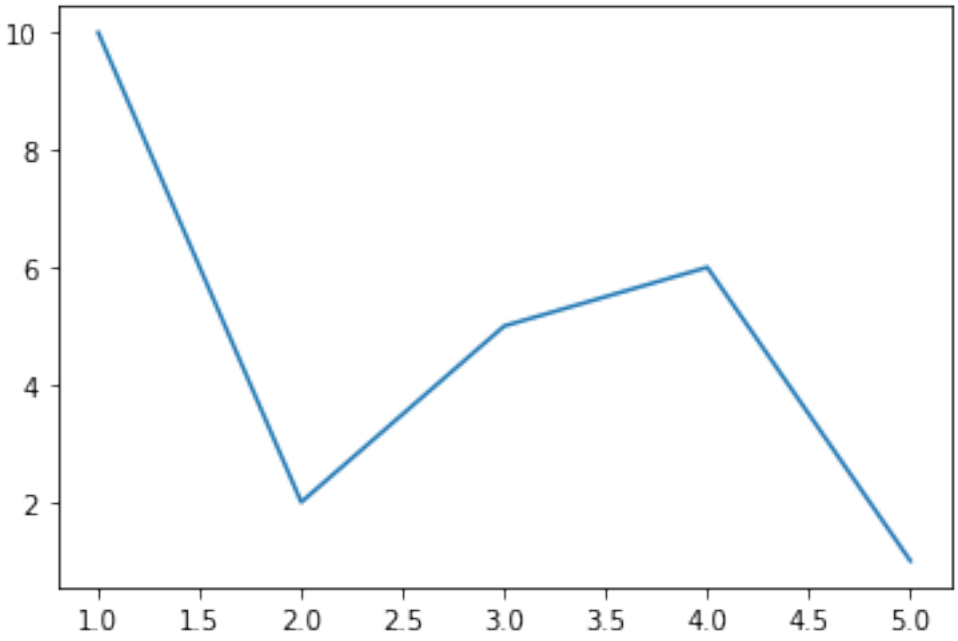
```
T = np.linspace(1,2,50)
Y = si.odeint(F,5,T)   # Y sous forme de colonne array

print("Les abscisses :",T)
print("Les solutions :",Y[:,0])
```

```
Les abscisses : [1.         1.02040816 1.04081633 1.06122449 1.08163265
1.10204082
 1.12244898 1.14285714 1.16326531 1.18367347 1.20408163 1.2244898
 1.24489796 1.26530612 1.28571429 1.30612245 1.32653061 1.34693878
 1.36734694 1.3877551  1.40816327 1.42857143 1.44897959 1.46938776
 1.48979592 1.51020408 1.53061224 1.55102041 1.57142857 1.59183673
 1.6122449  1.63265306 1.65306122 1.67346939 1.69387755 1.71428571
 1.73469388 1.75510204 1.7755102  1.79591837 1.81632653 1.83673469
 1.85714286 1.87755102 1.89795918 1.91836735 1.93877551 1.95918367
 1.97959184 2.        ]
Les solutions : [5.         5.06164252 5.12412632 5.18746042 5.25165342
5.31671412
 5.38265141 5.44947429 5.51719183 5.58581321 5.65534771 5.7258047
 5.79719363 5.86952406 5.94280564 6.01704814 6.09226141 6.1684554
 6.24564017 6.3238259  6.40302284 6.48324136 6.56449195 6.64678516
 6.73013171 6.8145424  6.90002814 6.98659995 7.074269   7.16304652
 7.25294389 7.34397259 7.43614422 7.5294705  7.62396329 7.71963453
 7.81649632 7.91456087 8.01384052 8.11434772 8.21609507 8.31909528
 8.42336121 8.52890584 8.63574228 8.74388378 8.85334373 8.96413565
 9.07627319 9.18977017]
```

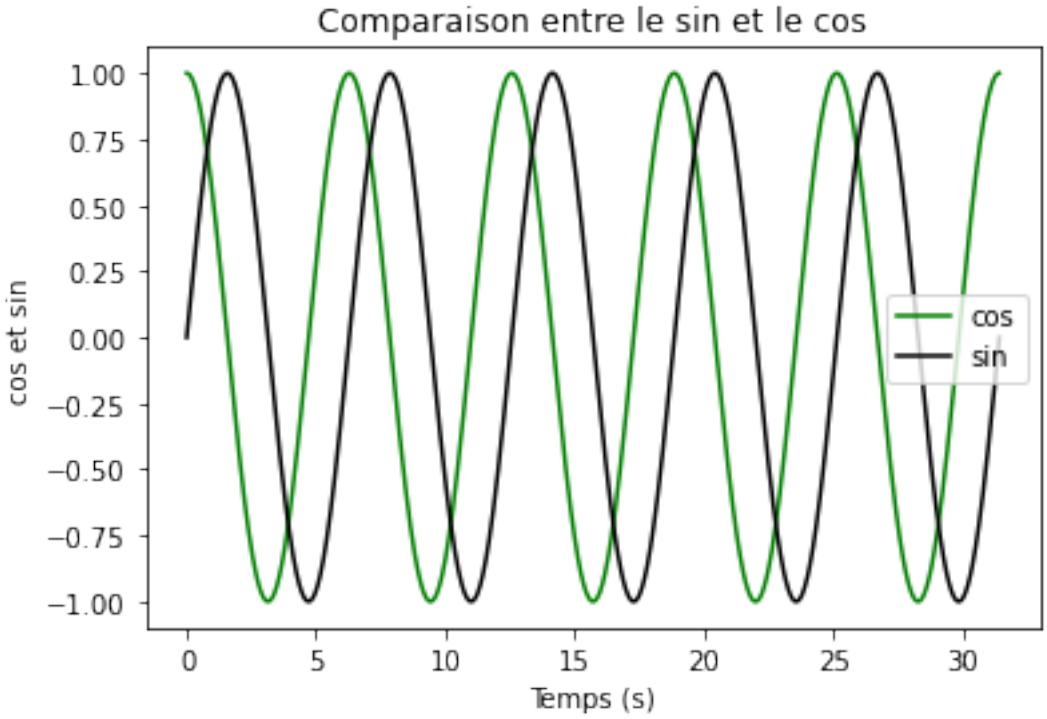
7 La bibliothèque matplotlib.pyplot

La bibliothèque matplotlib.pyplot nous permet de tracer des courbes

[ ]:
```
import matplotlib.pyplot as plt
L = [1,2,3,4,5]
P = [10,2,5,6,1]
plt.plot(L,P)
plt.show()
```



[ ]:
```
# Tracer le cos et le sin dans l'intervalle [0,10pi]
import numpy as np
import matplotlib.pyplot as plt
X=np.linspace(0,10*np.pi,1000)
Y=np.cos(X)
Z=np.sin(X)
plt.plot(X,Y,color="g",label='cos')
plt.plot(X,Z,color="k",label='sin')
plt.legend()
plt.title('Comparaison entre le sin et le cos')
plt.xlabel('Temps (s)')
plt.ylabel('cos et sin')
plt.show()
```


Comparaison entre le sin et le cos
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
cos et sin
0
5
10
15
20
25
30
Temps (s)
cos
sin