SPUS201 - UE SCIENCES : Introduction à la programmation 2

Denis Dubruel - Cours Magistral N°5

Année 2024/2025
courriel : prenom.nom@univ-cotedazur.fr

Table des Matières.

Table des matières

Modules

Sans autres indications, python ne sait pas faire de trigonométrie au démarrage !

```
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
```

Ajoutons :

```
>>> from math import cos
>>> cos(0)
1.0
```

Mais ... il en manque.

```
>>> sin(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sin' is not defined
```

Modules

Solution possible : Importer toutes les définitions du module.

```
>>> from math import *
>>> cos(0)
1.0
>>> sin(0)
0
```

Conserve les noms définis dans le module math.

⚠Ecrase des noms existants sans message !

```
>>> import math
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
```

```
>>> math.cos(0)
1.0
```

Utilise l'espace de nom du module. Pour utiliser la fonction il faut ajouter "math.".

Modules - espace de nom

Solution possible :

```
>>> cos = 3  # coefficient
>>>print(cos)
3
>>> from math import *
>>> print(cos)
<built-in function cos>
```

La variable cos a été détruite et le nom est utilisé pour la fonction. (équivalent à redéfinir cos par cos=math.cos)

```
>>> cos = 3  # coefficient
>>>print(cos)
3
>>> import math
>>> math.cos(0)
1.0
>>>print(cos)
3
```

Pas de conflit entre la variable cos et la fonction math.cos.

Modules - renommage

Possibilité de renommer un module ou une composante d'un module (fonction, classe, variable etc...)

```
>>>from math import cos as cosinus
>>>from math import sqrt as racine

>>>racine(2)
1.4142135623730951

>>>cosinus(0)
1.0
```

```
>>> import math as top

>>>top.cos(0)
>>>1.0

>>>top.sqrt(3)
>>>1.7320508075688772
```

Objectif : rester explicite pour avoir un code lisible !

Modules - importations

Ordre d'importation :

1. modules & bibliothèque standard :

   https://docs.python.org/fr/3.10/library/index.html)
2. les bibliothèques tierces https://pypi.org/

3. Vos modules.

Modules - importation

Les importations se placent en début de code (bonne pratique).

```
1 from math import cos, sin, sqrt, log10
2 import random
3 import os
4 from Mescodes   import  kiboguejamais , chargeFichier
5
6 ###
7 ###  Votre code qui utilise les fonctions importées.
8 ###
```

Modules - mise au point puis importation

Etape 1 : développement du code classiquement.

```python
from math import sqrt

def racine_ou_zero(x:float )-> float :
    """racine carrée ou renvoie 0"""
    try :
        resultat=sqrt(x)
        return resultat
    except ValueError:
        return 0

assert racine_carree_ou_zero(-1)==0
r2 : float = racine_carree_ou_zero(2)
assert  1.1414 < r2   and r2 < 1415
print("Le code a réussi les tests")
```

Modules - mise au point puis importation

Etape 2 :

Ex_modularisation.py

```
1  from math import sqrt
2
3  def racine_ou_zero(x:float )-> float :
4      """racine carrée ou renvoie 0"""
5      try :
6          resultat=sqrt(x)
7          return resultat
8      except ValueError:
9          return 0
10
11 if __name__=="__main__":
12         assert racine_carree_ou_zero(-1)==0
13         r2 : float = racine_carree_ou_zero(2)
14         assert  1.1414 < r2   and r2 < 1415
15         print("Le code a réussi les tests")
16 else :
17     print("Exécution du code du module.")
18     print(f'Dans le module __name__ = "{__name__}"')
```

Si **__name__ =__main__** alors le fichier est exécuté directement.

Si **__name__ =”nom du module”** alors il est importé comme un module dans un autre code.

Modules - mise au point puis importation

Etape 3 : Exécution du code Ex_test_module.py qui importe le module Ex_modularisation et renomme la fonction en rcz.

Ex_test_module.py

```
1 from   Ex_modularisation   import racine_carree_ou_zero as rcz
2
3 dix=rcz(100)
4
5 print(dix)
```

```
1 >>> %Run Ex_test_module.py
2 Exécution du code du module.
3 Dans le module __name__ = "Ex_modularisation"
4 dix = 10.0.
```

Quand rcz est appelée, on voit que la variable __name__ contient le nom du module, ainsi dans le test if du module Ex_modularisation éxécute la branche else.

Modules - mise au point puis importation

Résumons :

Ex_test_module.py

```
from   Ex_modularisation
import racine_carree_ou_zero
as rcz # importation

dix=rcz(100) # appel

print(dix)
```

```
>>> %Run Ex_test_module.py
Exécution du code du module.

Dans le module
__name__ = "Ex_modularisation"

dix = 10.0.
```

Ex_modularisation.py

```
from math import sqrt

def racine_ou_zero(x:float )-> float :
    """racine carrée ou renvoie 0"""
    try :
        resultat=sqrt(x)
        return resultat
    except ValueError:
        return 0

if __name__=="__main__":
        assert racine_carree_ou_zero(-1)==0
        r2 : float = racine_carree_ou_zero(2)
        assert  1.1414 < r2   and r2 < 1415
        print("Le code a réussi les tests")
else :
    print("Exécution du code du module.")
    print(f'Dans le module __name__ = "{__name__}"
```

Modules - Conclusion

Avantages des modules :

- **réutilisation du code** : ex. fonction pour charger les données d'un fichier *.csv
- **documentation :** dans le module (docstring """ cette fonction blabla .""")
- **développement à part**,uniquement sur le module.
- **tests intégrés** au module grâce à l'autotest if __name="__main__" :
- **espace de noms** distincts possible (avec nom.fonction)

A utiliser sans limite en TP.

Table des matières

Visualisation graphique

Matplotlib : très bien référencé, source ouverte, libre :

- https://matplotlib.org
- https://python.developpez.com/tutoriels/graphique-2d/matplotlib/

Exemple de visualisation graphique : **matplotlib** module Python, dédié aux représentations graphiques.

Génère des graphiques statiques, animés et interactifs. Génère des visualisations comme des courbes, des histogrammes, des graphiques en 3D, etc.

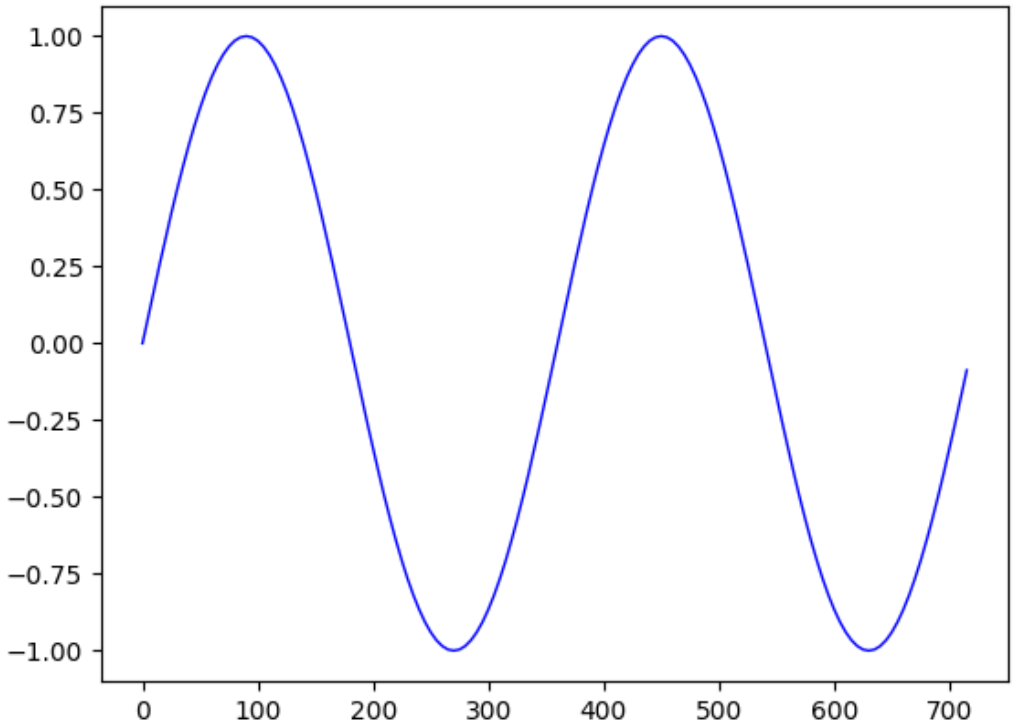
Préparation des données :

⚠ **même nombre** de valeurs en abscisse et en ordonnée ⚠

```python
### Préparation des paires de point (x,y)
import matplotlib.pyplot as plt
from math import cos,sin, pi

X : list[int]   = []    # abscisse
Y : list[float] = []    # première courbe
Yp : list[float] = []

for angle in range(0,720,5):
    X.append(angle)
    Y.append(sin(angle*pi/180)) # conversion degrés -> radians *pi/180

    Yp.append(cos(angle*pi/180)) # pour une 2eme courbe.
```

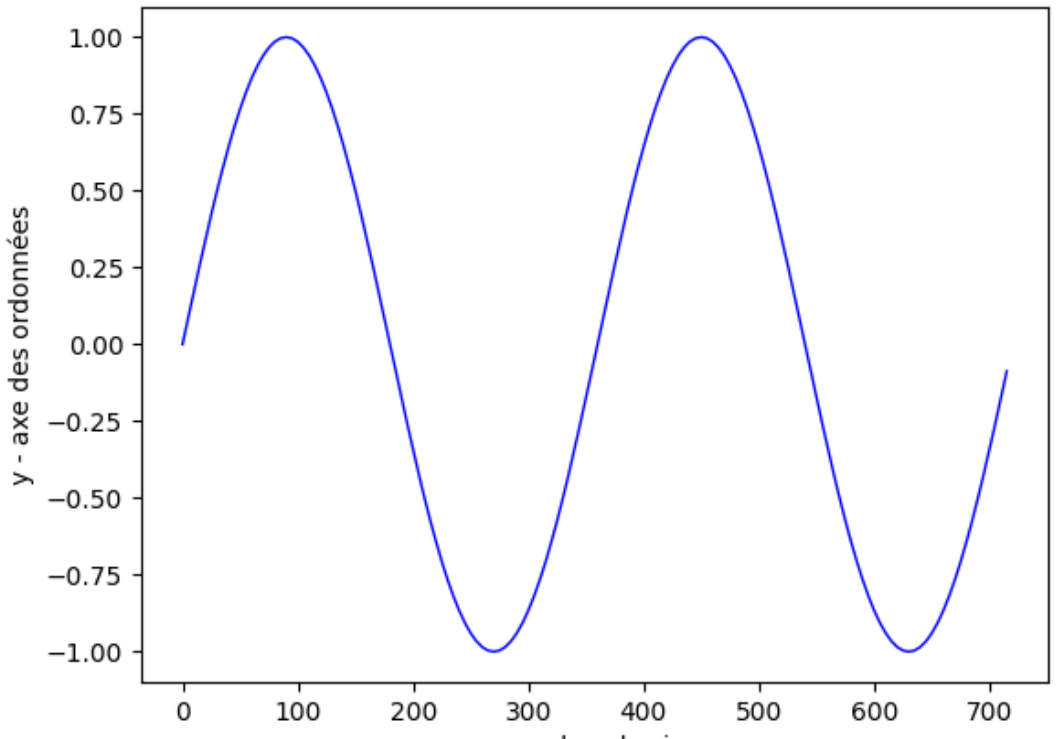
Visualisation graphique 1/6

```
# Tracé de la courbe représentant y en fonction de x avec le style
# styleDuGraphe, 'lépaisseur linewidth, le nom de la
# courbe à afficher dans la légende étant label
styleDuGraphe='b-'  # couleur bleue et trait continu
plt.plot(X, Y, styleDuGraphe, linewidth=1, label = "y = f(x)" )
plt.show()   # affiche le tracé
```



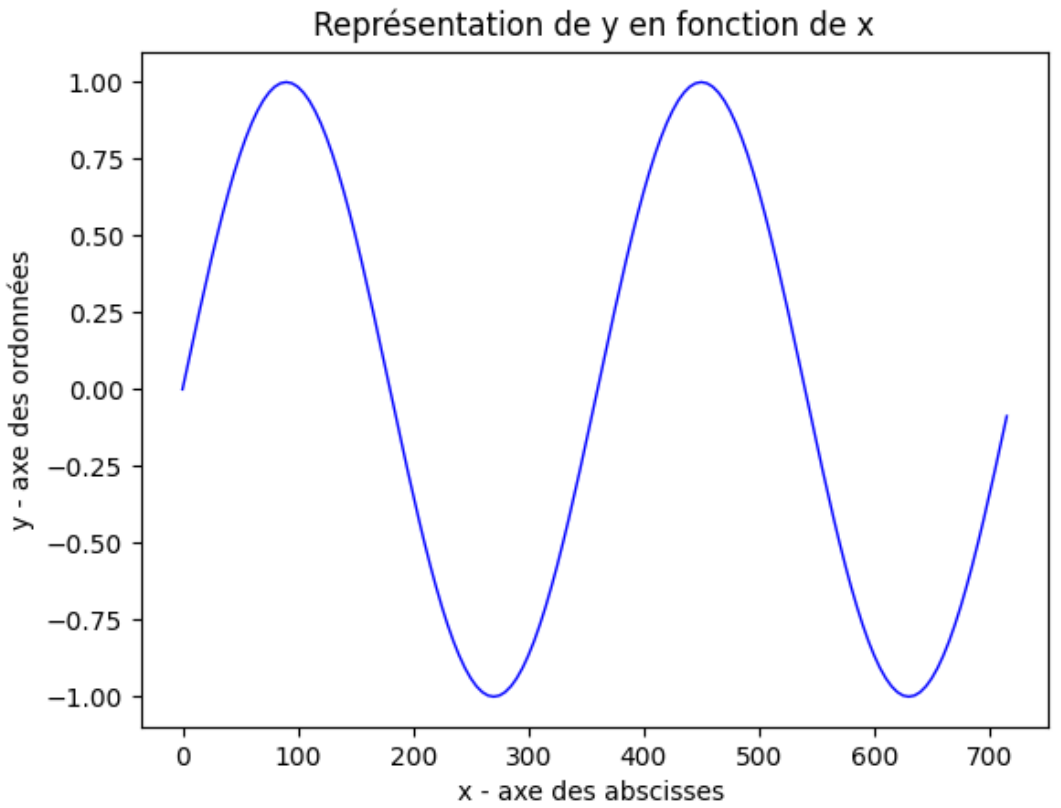
Visualisation graphique 2/6

```
1 # Ajouter des libellés sur les axes
2 styleDuGraphe='b-'
3 plt.plot(X, Y, styleDuGraphe, linewidth=1, label = "y = f(x)" )
4 plt.xlabel('x - axe des abscisses ') # ajout
5 plt.ylabel('y - axe des ordonnées')  # ajout
6 plt.show() # affiche le tracé
```



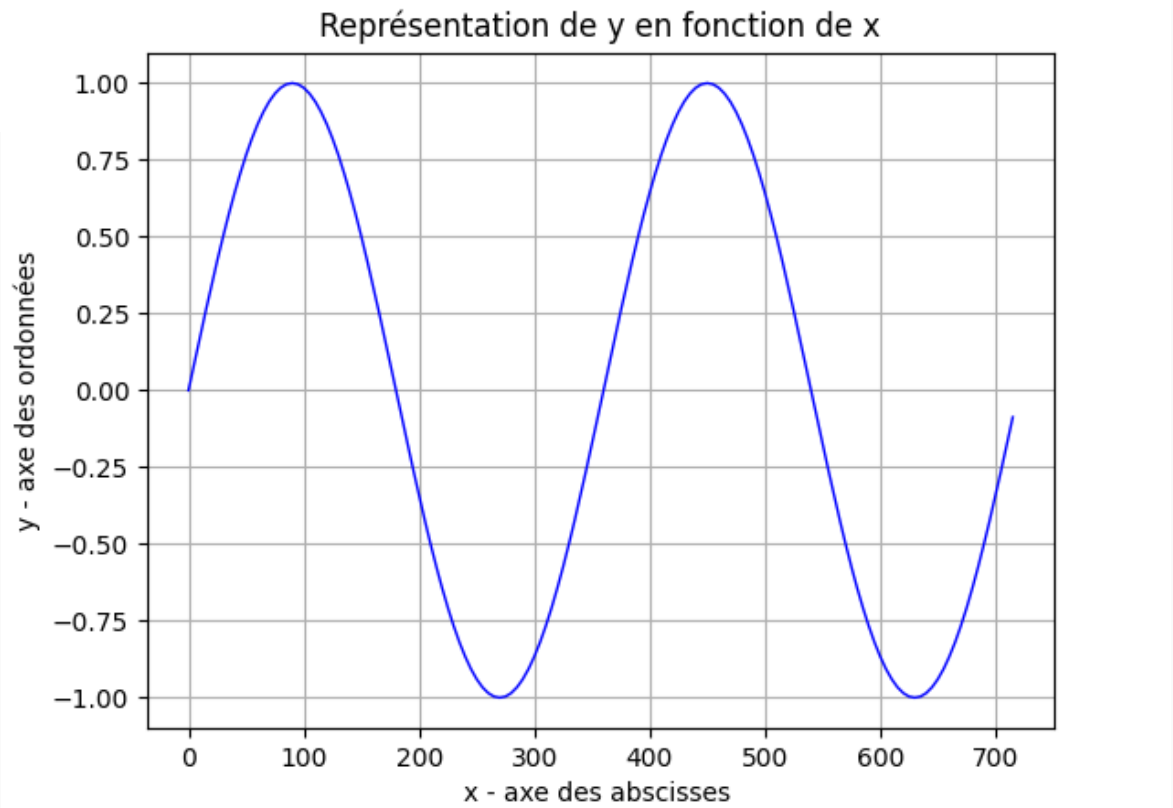
Visualisation graphique 3/6

```
1 #Ajouter un titre au graphique
2 styleDuGraphe='b-'
3 plt.plot(X, Y, styleDuGraphe, linewidth=1, label = "y = f(x)" )
4 plt.xlabel('x - axe des abscisses ')
5 plt.ylabel('y - axe des ordonnées')
6 plt.title('Représentation de y en fonction de x')   # ajout
7 plt.show()
```



Visualisation graphique 4/6

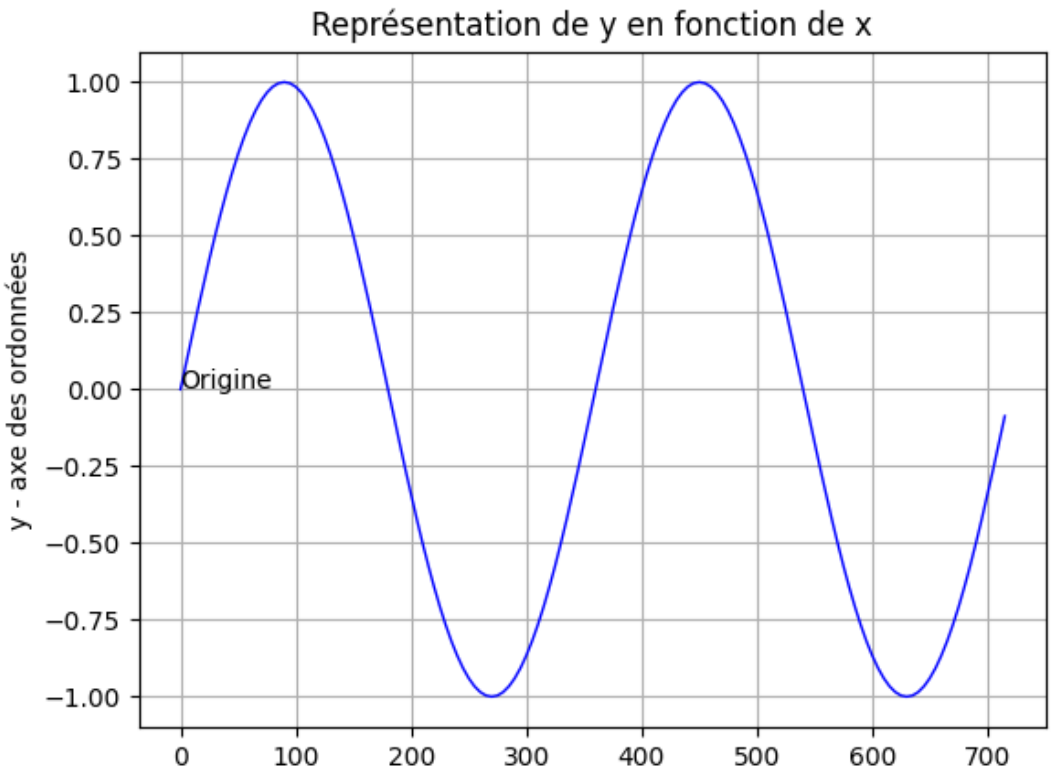
```
1 # Ajouter une grille au graphique
2 styleDuGraphe='b-'
3 plt.plot(X, Y, styleDuGraphe, linewidth=1, label = "y = f(x)" )
4 plt.xlabel('x - axe des abscisses ')
5 plt.ylabel('y - axe des ordonnées')
6 plt.title('Représentation de v en fonction de x')
7 plt.grid()    # ajout
8 plt.show()
```



Visualisation graphique 5/6

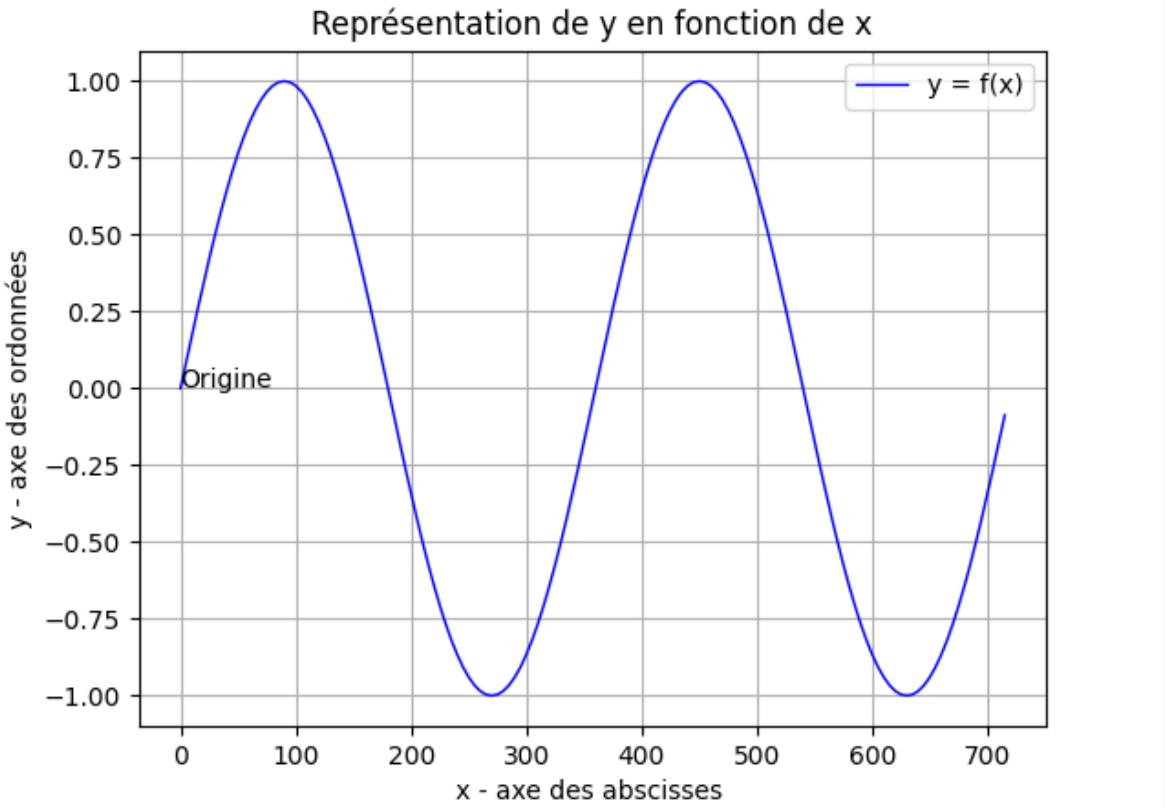
```
1 # Ajouter du texte dans le graphe à la position souhaitée
2 styleDuGraphe='b-'
3 plt.plot(X, Y, styleDuGraphe, linewidth=1, label = "y = f(x)" )
4 plt.xlabel('x - axe des abscisses ')
5 plt.ylabel('y - axe des ordonnées')
6 plt.title('Représentation de y en fonction de x')
7 plt.grid()
8 plt.text(0, 0, 'Origine') #
9 plt.show()
```

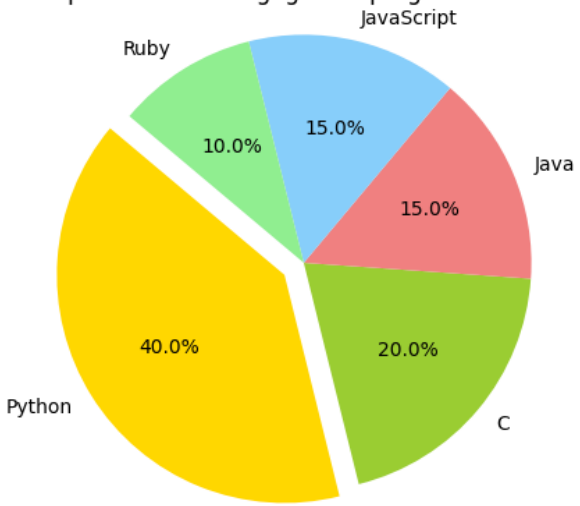
ligne 8 ajoutée !!

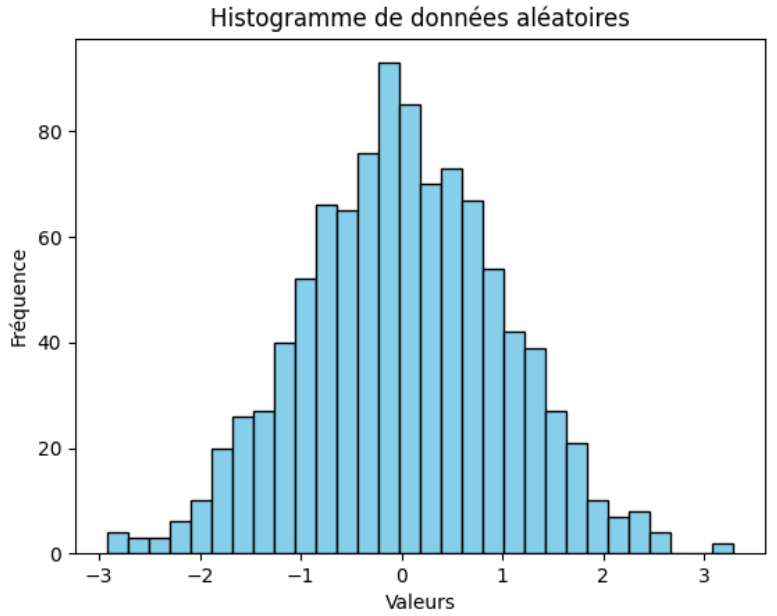


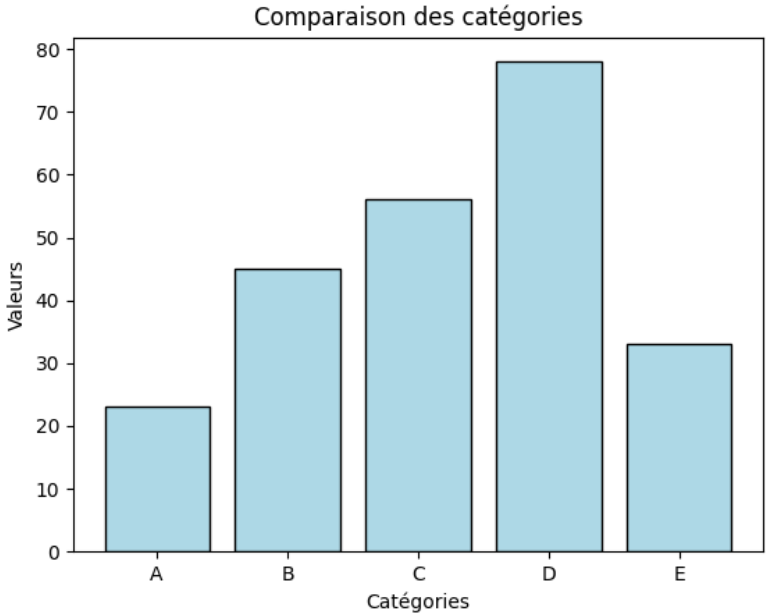
Visualisation graphique 6/6

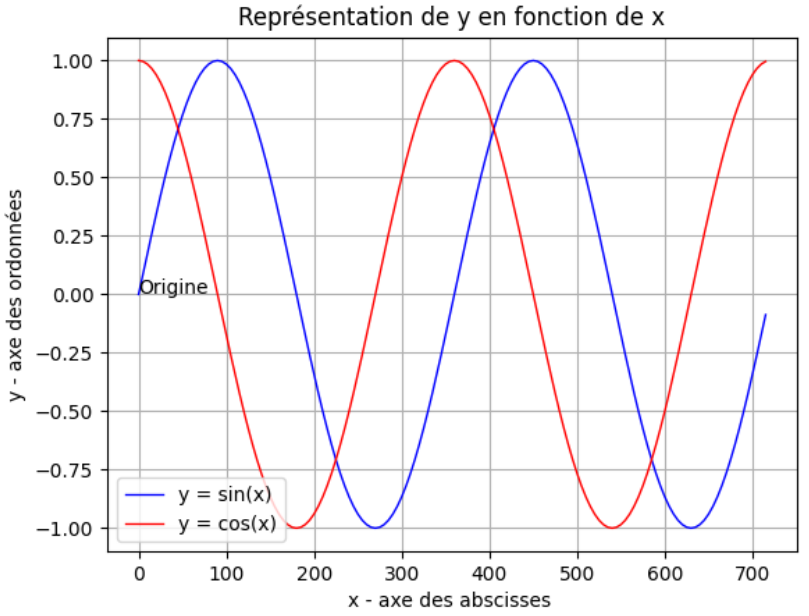
```
1  # Ajouter une légende avec le nom des courbes
2  styleDuGraphe='b-'
3  plt.plot(X, Y, styleDuGraphe, linewidth=1, label = "y = f(x)" )
4  plt.xlabel('x - axe des abscisses ')
5  plt.ylabel('y - axe des ordonnées')
6  plt.title('Représentation de y en fonction de x')
7  plt.grid()
8  plt.text(0, 0, 'Origine')
9  plt.legend() # ajout
10 plt.show()
```




Répartition des langages de programmation
JavaScript
Ruby
Java
Python
C
15.0%
10.0%
15.0%
40.0%
20.0%


Histogramme de données aléatoires
Fréquence
Valeurs
0
20
40
60
80
−3
−2
−1
0
1
2
3


Comparaison des catégories
Valeurs
Catégories
A
B
C
D
E
0
10
20
30
40
50
60
70
80


Représentation de y en fonction de x
y - axe des ordonnées
x - axe des abscisses
Origine
y = sin(x)
y = cos(x)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
100
200
300
400
500
600
700

Représentation graphique.

Matplotlib / Tableur ???

Matplotlib : rigoureux, précis, traitement reproductible car défini avec des lignes de codes.

Tableur : visuel, rapide, très bien pour quelques valeurs. Le travail fait avec souris et de nombreux clics...Comment s'en souvenir ??

Avantage Matplotlib : traitement de grand volume de données (ex en TP fichier de plus de 600 000 lignes, impossible à faire avec un tableur). Travail rigoureux et sérieux.

Représentation graphique.

Pour la visualisation graphique, les données peuvent venir de calculs précédents ou de données (sondages, statistiques, résultats numériques de tierces parties etc). A faire en TP : stastistique de votre prénom à l'état civil (source Insee,tableau de 686 539 lignes !!!)

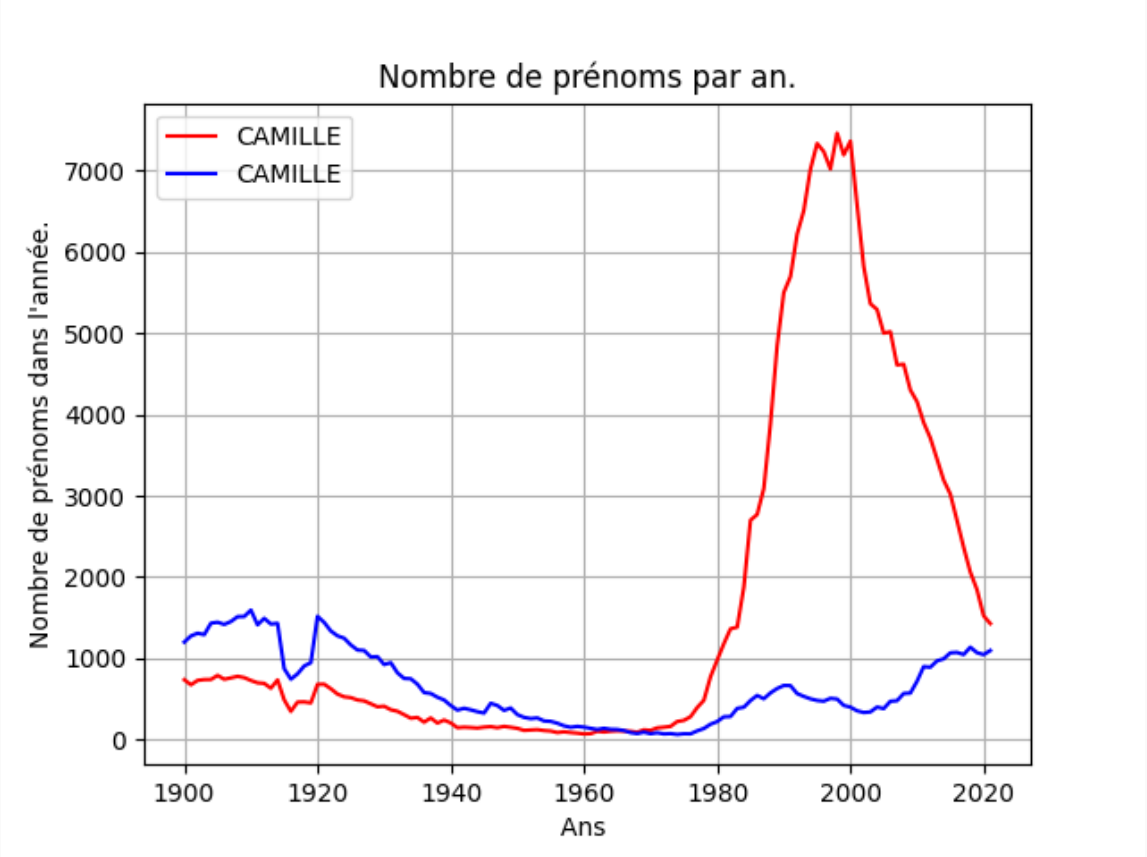


Table des matières

Numerical Python - Numpy

Nouveau module : Numerical Python ou Numpy en abrégé. Avantage, calculs bien plus rapide avec Numpy que sur des listes habituelles.

S'importe en général au début de code sous l'alias **np** :

```
1 import numpy as np
```

Numpy définit un nouveau type , les tableaux ou **array**. Très similaires aux listes mais beaucoup plus rapides pour les calculs.

Création à partir d'une liste :

```
1 import numpy as np
2 liste = [1,2,4]
3 tableau = np.array(liste)
4 print(tableau)
```

```
1 >>>
2  [1 2 4]
```

Possibilité de préciser le type des éléments :

```
1 >>> liste = [1,2,4]
2 tableau = np.array(liste , dtype=int)
3 print(tableau)
4 [1 2 4]
5 >>> liste = [1,2,4]
6 tableau = np.array(liste , dtype=float)
7 print(tableau)
8 [1. 2. 4.]
```

A partir d'une liste de liste => Matrice

```
liste_2 = [ [1,2,3], [4,5,6] ]
matrice = np.array(liste_2)
print(matrice)
```

```
>>>
[[1 2 3]
 [4 5 6]]
```

Avec précision du type si besoin :

```
>>> liste_2 = [ [1,2,3], [4,5,6]  ]
matriceFloat = np.array(liste_2 , dtype = float)
print(matriceFloat)
[[1. 2. 3.]
 [4. 5. 6.]]
```

Fonction zeros (que des zeros) / ones (à tester, que des 1!)

```
>>> print( np.zeros(4) )
[0. 0. 0. 0.]

>>> print( np.zeros(4 , dtype = "int") )
[0  0  0  0]
```

arange() génére une séquence de nombres séparés par un intervalle fixe dans un tableau :

```
>>>print( np.arange(5) )

 [0 1 2 3 4]

>>>print( np.arange(2, 10, 2) )

  [2 4 6 8]
```

Nombre d'élément d'un tableau :

```
1 t=np.arange(5)
2 print("Nombre d'éléments : ", t.size)   ### affichera  5
```

Réorganisation d'un tableau (1 lignes, 6 colonnes) en (3 lignes 2 colonnes)

```
1 tableau = np.array([3, 2, 5, 1, 6, 5])
2 tableau.shape = (3,2)
3 print(tableau)
4 print("dimensions du  tableau : ", tableau.shape)
```

```
1 >>>----------------------------------------------------------
2 [[3 2]
3  [5 1]
4  [6 5]]
5 dimensions du  tableau :  (3, 2)
```

Récupération des éléments d'un tableau (array) .Comme pour les listes !!

tableau[indiceDépart : indiceFin : pas]

```
1 tableau = np.array([0, 1 ,2,3,4,5,6,7,8,9,10,11,12,13,14 ])
2 extrait=tableau[2 : 14 :4]
3 print(extrait)
```

```
1 >>>
2  [ 2  6 10]
```

Extraction possible avec une boucle (comme une liste !!)

ou avec une liste d'indice :

```
1 >>> tableau[   [12 , 13 ]  ]    # avec une liste d'indice
2 [12  13]
```

Suppression d'éléments.

```
1 tableau = np.array([3, 2, 5, 1, 6, 5])
2 tableau = np.delete(tableau , [0 , 1 ]) # effacement des élément de rang 0 et 1 .
3 print(tableau)
```

```
1 >>>
2   [5 1 6 5]   # les éléments de rang 0 et 1 ont été effacés.
3   # le taille du tableau est modifiée également.
```

Un tableau (array) est un mutable donc se copie avec :

Sinon une modification de l'un modifie l'autre !!

```
1 tableau_2 = np.array(tableau_1)
2 # ou :
3 tableau_2 = tableau_1.copy( )
```

Numerical Python - Numpy

4 opérations sur les tableaux de **MÊME DIMENSION !!**

Tab1 + Tab 2 Tab1 - Tab2 Tab1 * Tab2 Tab1 / Tab2

Les opérations sont effectuées terme à terme (”même position dans le tableau”).

Tab1**2 élève au carré chaque terme du tableau.

cos(angle) calcule le cosinus pour chacun des termes.

etc. Ecriture très compacte.

Rem : Les opérations sur les matrices ne sont pas vues dans ce cours.

Exemple pour les quatres opérations. Terme à terme !

```
>>> tableau = np.array([0, 1 ,2,3,4,5,6 ])
print(f"tableau ={tableau}")
tableau =[0 1 2 3 4 5 6]

>>> somme   = tableau + tableau
print(f"somme ={somme}")
somme =[ 0  2  4  6  8 10 12]

>>> diff = tableau - tableau
print(f"diff ={diff}")
diff =[0 0 0 0 0 0 0]

>>> prod=tableau * tableau
print(f"prod ={prod}")
prod =[ 0  1  4  9 16 25 36]
```

Exemple pour les quatres opérations. Terme à terme !

```
>>>print(f"tableau ={tableau}")
tableau =[0 1 2 3 4 5 6]

>>> quotient =tableau / tableau
print(f"quotient ={quotient}")  # nan
<stdin>:1: RuntimeWarning: invalid value encountered in divide
quotient =[nan  1.  1.  1.  1.  1.  1.]
```

En cas d'erreur **nan** pour **n**ot **a** **n**umber. L'éxécution du code n'est pas interrompue, mais les futures valeurs ne seront plus calculables.

```
>>> print(quotient+quotient)   # ici quotient[0]=nan
[nan  2.  2.  2.  2.  2.  2.]
```

```
>>> cube=tableau**3   # idem pour les autres fonctions mathématiques.
print(f"cube ={cube}")
cube =[  0   1   8  27  64 125 216]
```

Comparaison de tableau. Renvoie un tableau de booléen. La comparaison se fait terme à terme.

Instruction	Description
greater()	Supérieur à
greater_equal()	Supérieur ou égal à
less()	Inférieur à
less_equal()	Inférieur ou égal à
equal()	Égal à
not_equal()	Différent de
logical_and()	Et logique
logical_or()	Ou logique

Exemple de comparaison de 2 tableaux :

```
>>> data1 =np.array([ [0.01 , 1.0] , [0.1 ,  1.1]])
    print(f"data1 ={data1}")

    data2 =np.array([ [0.02 , 2.0] , [0.2 ,  2.2]])
    print(f"data2 ={data2}")

    masque=np.greater(data2,data1)
    print(f"masque ={masque}")

data1 =[[0.01 1.  ]
        [0.1  1.1 ]]
data2 =[[0.02 2.  ]
         [0.2  2.2 ]]
masque =[[ True  True]
         [ True  True]]
```

Exemple de recherche dans 1 tableau :

```
>>> data =np.array( [1 , 2 ,8, 17 ,15, 6 ,1 ,3 , 25 ,5 ])
    print(f"data ={data}")

data =[    1   2      8     17     15      6     1     3   25     5]

>>>masque = np.logical_and(  2 < data ,  data < 16       )
   print(f"masque ={masque}")

masque =[False False  True False  True  True False  True False  True]

>>>valeurs = data[masque]
   print(f"valeurs ={valeurs}")

valeurs =[ 8 15  6  3  5]
```

Numerical Python - Numpy

Conclusion : Matplotlib et Numpy sont 2 modules pour traiter précisément et rigoureusement des données de différentes origines. Ces outils sont très bien documentés :

- https://matplotlib.org/
- https://numpy.org/

En python d'autre outils existent pour le traitement des données :

Panda (https://pandas.pydata.org/,

Seaborn (https://seaborn.pydata.org/)

Licence

Ce document est publié sous licence Creative Commons :

- ©2024 — Denis Dubruel - Université Côte d'Azur
- Attribution
- Utilisation non commerciale
- Partage dans les mêmes conditions 4.0 International

https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.fr