Corrigé Code de partage avec Capytale : d8e9-1683377

L'objectif de ces séances est de récapituler l'ensemble des savoir-faire de l'année.

1. Les commandes prédéfinies

   Après avoir importé `numpy` pour la plupart des commandes.

Ecriture mathématique	π	e	$\ln(5)$	e^3	$\sqrt{11}$
Ecriture avec Python	`np.pi`	`np.e`	`np.log(5)`	`np.exp(3)`	`np.sqrt(11)`

Ecriture mathématique	$\lvert -8\rvert$	$\leqslant$	$\geqslant$	$\neq$
Ecriture avec Python	`np.abs(-8)`	`<=`	`>=`	`!=`

2. Afficher un résultat

   Si on écrit un programme dans l'éditeur et qu'on l'exécute, il est possible qu'il n'affiche rien (par exemple le calcul de $f(2)$ avec f une fonction prédéfinie). Quelle commande permet l'affichage d'un résultat ?

   C'est la commande `print`

3. Demander une information

   Ecrire un programme qui demande un entier à l'utilisateur et renvoie son carré.

   Il s'agit de la commande `input`. De plus que la donnée entrée soit comprise par Python comme un nombre. Pour cela, on utilise `float` pour transformer l'information entrée en un nombre réel (`int` pour un nombre entier).

```
x=float(input("entrer un nombre"))
print(x**2)
```

4. Boucle for

   Ecrire un programme qui affiche le logarithme et la racine carrée des 10 premiers entiers.

```
import numpy as np
for i in range(1,11):
    print(np.log(i),np.sqrt(i))
```

5. Listes

   Définir deux listes qui affichent les mêmes résultats que le programme précédent, puis une troisième qui affiche le quotient des termes des deux listes deux à deux.

```
import numpy as np

L1=[np.log(i) for i in range(1,11)]
L2=[np.sqrt(i) for i in range(1,11)]
L3=[L1[i]/L2[i] for i in range(0,10)] # attention aux indices
```

6. Boucle while

   Ecrire un programme permettant de déterminer le premier entier n tel que $e^{-\sqrt{n}} \leqslant 10^{-4}$

```
import numpy as np

n=0
while np.exp(-np.sqrt(n))>10**(-4):
    n=n+1
print(n)
```

7. Calcul et représentation d'une suite

Calculer et représenter (avec des points) les 100 premiers termes de la suite u définie par :

$$\begin{cases} u_1 = 1 \\ \forall n \in \mathbb{N}^*, \quad u_{n+1} = 1 - e^{-u_n} \end{cases}$$

Faire une deuxième représentation à l'aide d'un diagramme en bâtons.

Pour pouvoir représenter, il faut garder en mémoire les valeurs, on va donc créer une liste de valeurs, de manière récursive en utilisant une boucle `for`.
Puis la commande `plt.plot(u)` permet de représenter la suite et `x` de définir les abscisses adéquates (par défaut, cela serait $0, 1, 2, \ldots, 100$). On peut modifier avec `'+'` pour afficher des points plutôt qu'une ligne brisée. Enfin `plt.bar` permet d'afficher le diagramme en bâtons.

```
import numpy as np

u=1
L=[1] # on créée une liste pour garder
    les valeurs
for i in range(2,101):
    u=1-np.exp(-u)
    L.append(u)

import matplotlib.pyplot as plt

x=np.arange(1,101,1) # pour commencer les
    abscisses à 1
plt.plot(x,L,'+') # variante plt.bar(x,L)
plt.show()
```

8. Définir une fonction

Définir avec Python la fonction f, définie sur $\mathbb{R}_+^*$ par : $f(x) = x\ln(x)$

```
import numpy as np
def f(x):
  return np.log(x)
```

9. Utiliser une condition

Modifier le programme précédent pour qu'il définisse la fonction f définie sur $\mathbb{R}$ par :

$$\forall x \in \mathbb{R}, \quad f(x) = \begin{cases} x\ln(x) & \text{si} \quad x > 0 \\ x^2 & \text{si} \quad x \leqslant 0 \end{cases}$$

```
import numpy as np
def f(x):
  if x>0 :
    return np.log(x)
  else :
    return x**2
```

10. Représentation graphique

Représenter la fonction précédente sur l'intervalle $[-10, 10]$, émettre une hypothèse sur la continuité de la fonction.
Python peut présenter une difficulté pour la représentation de ce type de fonction. Auquel cas, on construira plus progressivement la liste des ordonnées.

Avec la méthode classique, Python affiche un message d'erreur. Il n'arrive pas à calculer une liste d'images quand la fonction comporte une condition : `y=f(x)` avec `x` qui contient plusieurs valeurs (ici on pourrait logiquement définir `x=np.linspace(-10,10,100)`). Il faut donc trouver une parade, soit en définissant deux fonctions (ainsi que deux listes d'abscisses et deux listes d'ordonnées), soit en calculant les images une par une, ce qui est proposé ci-dessous.

```
import matplotlib.pyplot as plt
x=np.linspace(-10,10,100)
y=[f(x[i]) for i in range(0,100)]
plt.plot(x,y)
plt.show()
```

11. Trouver une valeur approchée à l'aide de la méthode de dichotomie

Sachant que, pour $n \in \mathbb{N}^*$, l'équation $x^n + x - 1 = 0$ admet une unique solution strictement positive notée u_n et que $u_n \in]0,1[$ pour tout $n \in \mathbb{N}^*$, compléter la fonction Python suivante renvoyant, pour n donné en entrée, une valeur approchée u_n à 10^{-3} près

```
def valeur_approchee(n):
  a=0
  b=1
  while b-a>10**(-3) :
    c=(a+b)/2
    if (c**n+c-1)>0 :
      b=c
    else :
      a=c
  return c
```

Comme d'habitude avec l'algorithme de dichotomie, c'est la condition sur la largeur de l'intervalle qui dicte l'arrêt (ou non) de la boucle. Ensuite, on peut comprendre avec $f(0)$ (qui est négatif) et $f(1)$ (qui est positif) que lorsque le résultat de $f(c)$ est positif, l'intervalle se resserre à droite et inversement.
Enfin on peut demander à la fonction de renvoyer le dernier milieu calculé.

12. Définir et exploiter une matrice

Avec Python définir la matrice $A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$

Calculer A^2 et $I_4 + A + A^2 + A^3$ et interpréter les résultats.

Pour l'interprétation, on va considérer qu'il s'agit de la matrice d'adjacence d'un graphe (non orienté car la matrice est symétrique).
Avec A^2, on obtient la matrice des chaines de longueur 2 reliant toute paire de sommets.
Et la matrice $I_4 + A + A^2 + A^3$ permet de tester le critère de connexité. En exécutant le programme, on trouve une matrice dont tous les coefficients sont strictement positifs, ce qui signifie que le graphe est connexe.

```
import numpy as np
import numpy.linalg as al

# on définit la matrice
M=np.array([[0,1,1,0],[1,0,0,1],[1,0,0,0],[0,1,0,0]])

# on calcule M^2
np.dot(M,M)
# variante avec al.matrix_power(M,2)

# matrice de connexité
M_C=np.zeros(4) # ne contient que des zéros au début (matrice 4,4)
for k in range (0,4):
```

```
    M_C=M_C+al.matrix_power(M,k) # on complète en ajoutant les puissances (
entre 0 et 3) de M
```

13. Générer un nombre aléatoire sur une plage donnée

    Ecrire une commande qui renvoie de manière aléatoire et équiprobable un nombre entier entre 1 et 10.

    Il s'agit simplement de la commande `randint`. Attention avec la bibliothèque `numpy.randint`, le deuxième nombre est exclu.
    Donc ici, après avec importé la bibliothèque avec `import numpy.random as rd` on rentrera la commande `rd.randint(1,11)`

14. Simuler une variable aléatoire suivant une loi de référence

    Deux joueurs s'affrontent dans un jeu de PILE ou FACE avec une même pièce donnant PILE avec une probabilité $p \in]0,1[$ selon le protocole suivant : les deux joueurs lancent la pièce jusqu'à obtenir PILE. Celui qui fait le moins de tirages gagne. Si les deux joueurs ont effectué le même nombre de lancers pour obtenir PILE, ils recommencent, et ce jusqu'à ce qu'un des deux joueurs soit déclaré vainqueur.
    Compléter la fonction Python ci-contre permettant de simuler le nombre de lancers nécessaires avant qu'un des deux joueurs ne soit déclaré vainqueur, avec une valeur de p donnée en entrée.

```
import numpy.random as rd

def simul(p):
  x=rd.geometric(p)
  y=rd.geometric(p)
  while x==y:
    x=rd.geometric(p)
    y=rd.geometric(p)
  N=min(x, y)
  return N
```

    Les rangs d'apparition du premier pile pour chacun des deux joueurs (`x` et `y` du programme) suivent chacun une loi géométrique de paramètre p et il suffit donc d'utiliser la commande `rd.geometric` pour simuler une telle loi. La boucle `while` sert simplement à régler les cas d'égalité et on ne fait que « rejouer » dans ce cas.
    A la fin, on veut connaitre le plus petit des deux résultats, il suffit donc de renvoyer `N`
    Pour tester la fonction, on peut prendre des valeurs de p « extrêmes », par exemple `simul(0.01)`