TP4: listes, courbes

Le document de référence doit être apporté en TP. Les exercices 1 à 10, incontournables, doivent être parfaitement traités en séance de TP. Les exercices suivants peuvent être traités en fin de séance ou à la maison, pour s'entraîner. Tous ces exercices sont susceptibles d'être posés en DS.

Créer un dossier "TP4" dans votre répertoire personnel. Pour l'exercice 1, dans Pyzo, enregistrer un fichier nommé "exo1.py" dans le dossier "TP4". Pour l'exercice 2, dans Pyzo, enregistrer un fichier nommé "exo2.py" dans le dossier "TP4"..... Pour chaque exercice, le fichier python sera régulièrement sauvegardé et éxécuté intégralement avec la commande "Run file as script" du menu Run (Ctrl+Shift+E). Les instructions d'affichage seront saisies sur le fichier python. Au fur et à mesure de l'avancée de l'exercice, les instructions devenant inutiles et gênantes pourront être désactivées en utilisant le caractère #.

Exercice 1. *Dans cet exercice, le terme tableau désigne une liste.*

1. *Ecrire une fonction nommée "argmin" ayant pour argument t (tableau de nombres) et retournant un indice j tel que $t[j]$ est minimum. (On utilisera la variante "initialisation avec un premier terme".) Faire afficher argmin([7,2,6,4,9,3,0,8,5,1]).*
2. *Considérons t une liste de longueur n. Déterminer, en fonction de n, le nombre de comparaisons $<$ effectuées lors de l'éxécution de $argmin(t)$.*

Remarque.

Pour un tableau t de longueur n, le nombre d'opérations effectuées lors de l'éxécution de $argmin(t)$ est de l'ordre de n. On dit donc que la complexité de l'algorithme est linéaire en ordre de grandeur.

Exercice 2. *Dans cet exercice, le terme tableau désigne une liste.*

La distance entre deux nombres x et y est $|x-y|$. En python, si x est un nombre, $abs(x)$ est la valeur absolue de x. En python, $float("inf")$ désigne le flottant $+\infty$.

1. *Ecrire une fonction "argmindist" ayant pour argument t (tableau de nombre) et retournant (l,k) un couple d'indices avec $l < k$ tel que la distance entre $t[l]$ et $t[k]$ est minimum. (On utilisera la variante "initialisation sans premier terme".) Afin de considérer les uns après les autres les couples d'indices (j,i) tels que $j < i$, on effectuera deux boucles imbriquées, en choisissant la variable i pour la boucle externe.*
   Faire afficher argmindist([5,20,50,30,10,36,17,1,45]).
2. *Considérons t une liste de longueur n. On éxécute $argmindist(t)$.*
   (a) *Déterminer, en fonction de i, le nombre d'opérations $<$ effectuée lors d'une éxécution de la boucle interne.*
   (b) *Déterminer, en fonction de n, le nombre de comparaisons $<$ effectuées lors de l'éxécution de $argmindist(t)$*

Remarque.

Pour une liste t de longueur n, le nombre d'opérations effectuées lors de l'éxécution de $argmindist(t)$ est de l'ordre de n^2. On dit donc que la complexité de l'algorithme est quadratique en ordre de grandeur.

Tableaux numpy 1D
Les tableaux numpy 1D ressemblent à des listes.
import numpy as np
$x = np.array([x_0, x_1, ..., x_{n-1}])$
(affecter à x le tableau numpy 1D de termes $x_0, x_1, ..., x_{n-1}$)
$y = np.array([y_0, x_1, ..., y_{n-1}])$
(affecter à y le tableau numpy 1D de termes $y_0, y_1, ..., y_{n-1}$)
Tracés de lignes brisées
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.axis("equal")
plt.show()
(tracer la ligne brisée joignant les points de coordonnées (x_0, y_0),...,(x_{n-1}, y_{n-1}))

Exercice 3. *Tracer le triangle dont les sommets ont pour coordonnées* $(1, 0)$, $(0, 2)$ *et* $(-1, -1)$.

Tracer une courbe comme une ligne brisée
import numpy as np
$x = np.linspace(a, b, n)$
(affecter à x le tableau numpy $1D$ dimension de termes $x_0, ..., x_{n-1}$
avec $a = x_0 < x_1 < ... < x_{n-1} = b$ et régulièrement espacés, l'espacement étant donc $\frac{b-a}{n-1}$).
$y = np.exp(x)$
(affecter à y le tableau numpy de termes $exp(x_0)$,...$exp(x_{n-1})$)
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.axis("equal")
plt.show()
De même avec les fonctions log, $cosh$, $sinh$, cos, sin, tan, $acos$, $asin$, $atan$, $fabs$, $sqrt$.

Exercice 4. *Tracer la courbe d'équation* $y = exp(x)$ *sur* $[-1, 1]$ *en se basant sur* 100 *points.*

Exercice 5. *Effectuer l'instruction : from math import pi.*
Tracer la courbe d'équation $y = cos(x)$ *sur* $[-pi, pi]$ *en se basant sur* 100 *points.*

Opérations termes à termes sur deux tableaux numpy numériques de même format
Soit s un tableau numpy 1D de termes numériques s_0,....,s_{n-1}.
Soit t un tableau numpy 1D de termes numériques $t_0, ..., t_{n-1}$.
$s + t$ est le tableau numpy 1D de termes numériques $s_0 + t_0$,....,$s_{n-1} + t_{n-1}$
De même avec les opérations $-$, $*$, $/$, $**$

Exercice 6. *Tracer la courbe d'équation* $y = cos(x) + sin(x)$ *sur* $[-pi, pi]$ *en se basant sur* 100 *points.*

Opérations termes à termes sur un tableau numpy numérique
Soit x un tableau numpy de termes numériques x_0,....,x_{n-1}.
Soit c un nombre
$x + c$ est le tableau numpy de termes $x_0 + c, x_1 + c, ..., x_{n-1} + c$
De même avec les opérations $-$, $*$, $/$, $**$.

Exercice 7. *Tracer la courbe d'équation* $y = x^2 + 3x + 5$ *sur* $[0, 1]$ *en se basant sur* 100 *points.*

Exercice 8. *Tracer la courbe d'équation* $y = cos(2x)$ *sur* $[-pi, pi]$ *en se basant sur* 100 *points.*

Tracés superposés sur une figure

```
import matplotlib.pyplot as plt
plt.plot(...,...)
plt.plot(...,...)
plt.axis("equal")
plt.show()
```

Tracés sur des figures séparées

```
import matplotlib.pyplot as plt
plt.figure()
plt.plot(...,...)
plt.axis("equal")
plt.figure()
plt.plot(...,...)
plt.axis("equal")
plt.show()
```

Exercice 9. *Tracer sur une même figure les courbes d'équations* $y = cos(x)$ *et* $y = sin(x)$ *sur* $[-pi, pi]$ *en se basant sur* 100 *points.*

Exercice 10. *Tracer sur des figures séparées les courbes d'équations* $y = cos(x)$ *et* $y = sin(x)$ *sur* $[-pi, pi]$ *en se basant sur 100 points.*

Exercice 11. *Une liste de nombres* $l = [a, b, c, d, ...]$ *est dite triée ssi* $a \leq b \leq c \leq d \leq ....$ *Ecrire une fonction "testcroi" ayant pour en argument une liste* l *d'entiers et retournant True si la liste* l *est croissante et False sinon. Faire afficher testcroi(*$[1, 3, 4, 9]$*), testcroi(*$[3, 1, 4, 9]$*), testcroi(*$[1, 3, 9, 4]$*), testcroi(*$[1, 9, 3, 4]$*).*

Exercice 12. *Ecrire une fonction "seleclis" ayant pour arguments* l *une liste de flottants et* a *et* b *deux flottants tels que* $a \leq b$ *et retournant la liste constituée des termes de* l *compris entre* a *et* b *(au sens large). Faire afficher seleclis(*$[1.3, 2.9, 3.5, 8.4, 4.5, 0.9, 3.8], 1.5, 4.2$*).*

Exercice 13. *Ecrire une fonction "lisfacto" ayant pour argument un entier naturel* n *et retournant la liste de termes* $0!,1!,2!,...,n!$*. Faire afficher* $lisfacto(7)$*.*

Exercice 14.

1. *Ecrire une fonction "nbocc" ayant pour arguments une liste* l *et un objet* x *et retournant le nombre de termes de la liste égaux à* x*. Faire afficher nbocc(*$[2, 2, 1, 3, 2, 1, 4], 1$*).*
2. *En déduire une fonction "majoritaire" ayant pour argument une liste* l *et retournant un terme de la liste dont le nombre d'occurences est maximum. Faire afficher majoritaire(*$[2, 2, 1, 3, 2, 1, 4]$*).*

Exercice 15.

1. *Ecrire une fonction "apparli" ayant pour arguments une liste* l *et un objet* x *et retournant True si* x *est un terme de* l *et False sinon.*

   Faire afficher $apparli([2, 1], 3)$ *et* $apparli([2, 1, 3], 2)$*.*
2. *La liste* $[2, 2, 1, 3, 2, 1, 4]$ *possède des répétitions (*2 *est répété* 3 *fois et* 1 *est répété* 2 *fois). En supprimant les répétitions, on obtient la liste* $[2, 1, 3, 4]$*. Ecrire une fonction "sansrepet" ayant pour argument une liste* l *en renvoyant une liste* m *obtenue en supprimant les éventuelles répétitions dans* l*. Faire afficher sansrepet(*$[2, 2, 1, 3, 2, 1, 4]$*).*