TP 1

Normalement, vous devez connaître le langage python, l'environnement pycharm & le développement par les tests. Si ce n'est pas le cas, les tutos suivants sont pour vous :

- https://informatique.centrale-marseille.fr/tutos/post/python-bases.html
- https://informatique.centrale-marseille.fr/tutos/post/utilisation-pycharm-bases.html
- https://informatique.centrale-marseille.fr/tutos/post/python-tests.html

Calcul de Puissance

On cherche à calculer x^y

Itératif et récursif naïf

Codez un algorithme itératif et un algorithme récursif naïf permettant de calculer la puissance de deux entiers et leurs tests associés.

Exponentiation rapide

Coder l'algorithme d'exponentiation rapide (cf TD 1, Exercice 5) et ses tests.

Calcul de complexité

Pour mesurer ce temps on pourra utiliser la méthode process_time du module time de python (si votre python3 est vieux, utilisez la méthode clock de time).

```
import time

temps_depart = time.process_time()
#ce que l'on veut mesurer
delta_temps = time.process_time() - temps_depart
```

Vérifiez que :

- le temps pris par l'algorithme itératif augmente suivant la valeur de y,
- le temps mis par l'algorithme itératif et rapide ne dépend pas de x,
- le rapport entre le temps mis pour résoudre une exponentiation avec l'algorithme rapide et celui mis avec l'algorithme naïf tend vers 0.

Mesurer précisément le temps mis pour exécuter un algorithme est compliqué. Les oscillations sont normales car le système, l'ide et même python peuvent faire des choses en parallèle. La mesure de temps utilisée n'est donc pas rigoureusement

proportionnelle à la complexité de l'algorithme mais en est une bonne approximation.

Affichage de courbes

Utilisez le code ci-dessous pour afficher une courbe avec matplotlib où l'abcisse est y et l'ordonnée le temps mis pour calculer x^y.

```
import matplotlib.pyplot
from math import log


coordonnees_abcisses = range(2, 101)

x_fois_2 = []
x_carre = []
x_log_x = []

for x in coordonnees_abcisses:
    x_fois_2.append(x * 2)
    x_carre.append(x * x)
    x_log_x.append(x * log(x))

matplotlib.pyplot.ylabel("axe des ordonnees")
matplotlib.pyplot.xlabel("axe des abcisses")

matplotlib.pyplot.plot(coordonnees_abcisses, x_fois_2, color="#ff0000")
matplotlib.pyplot.plot(coordonnees_abcisses, x_carre, color="#00ff00")
matplotlib.pyplot.plot(coordonnees_abcisses, x_log_x, color="#0000ff")

matplotlib.pyplot.show()
```

Supperposez les courbes pour les 3 algorithmes. Conclusions ?

Corrélation aux complexités théoriques

Le rapport entre le temps mis pour effectuer un algorithme et sa complexité théorique doit être une droite. Utilisez numpy et polyfit pour calculer la droite de régression linéaire de ce rapport pour l'algorithme naïf et l'algorithme rapide.

```
import numpy

x = list(range(10))
y = [5 * i + 10 for i in x]  # y = 5x + 10

# régression linéaire
```

```
a, b = numpy.polyfit(x, y, 1)
print(a, b)  # a = 5, b = 10
```

En déduire le coefficient de corrélation linéaire (on pourra utiliser la méthode corrcoef de numpy). Conclusion ?

Up : accueil