1 TP bonus : cribler les nombres premiers

Notions pratiquées : nombres premiers, listes, boucles, module time

Dans ce TP on se propose de générer la liste des nombres premiers inférieurs à une limite donnée, de différentes manières, et de comparer le temps nécessaire selon l'algorithme choisi.

1.1 Tester si un nombre est premier

- La première méthode est simple et a déjà été réalisée en TD. Il s'agit de tester tous les diviseurs possibles entre 2 et n. Si le seul diviseur trouvé est n lui-même, alors on conclut que n est premier, sinon il ne l'est pas.
- Écrire une première fonction `testPremier(n)` qui vérifie si n est premier en testant tous les diviseurs entre 2 et n.
- On pourra déjà améliorer un peu cette méthode en arrêtant avant n : quel est le plus grand diviseur qu'il est nécessaire de tester ?

1.2 Générer une liste de nombres premiers

On veut maintenant générer la liste de tous les nombres premiers inférieurs ou égaux à une certaine borne donnée.

- Écrire une fonction `genererPremiers(sup)` qui reçoit un entier *sup*, et calcule et renvoie la liste des nombres premiers inférieurs ou égaux à *sup*. Il faut donc tester chaque nombre avec notre fonction `testPremier` avant de l'ajouter à la liste.
- Tester la fonction en générant la liste des nombres premiers jusqu'à 100,
- Si cela fonctionne, tester maintenant en générant les nombres premiers jusqu'à 20000. Que remarque-t-on ?

1.3 Cribler les nombres premiers

L'intuition pour améliorer encore notre recherche des nombres premiers, consiste à réaliser qu'on n'a besoin de tester que les diviseurs eux-mêmes premiers. Par exemple si on a déjà testé que n n'est pas divisible par 3, alors il sera inutile d'essayer de le diviser par 9.

- Écrire une fonction `criblerPremiers(`*lim*`)` qui utilise cette idée : la fonction génère la liste des nombres premiers jusqu'à *lim*, mais en ne testant que les diviseurs premiers de ce nombre, et en s'arrêtant dès que possible.
- Dans un premier temps, afficher les différentes étapes du calcul: quel nombre est testé pour savoir s'il est premier, quel diviseur de ce nombre est testé, quels nombres premiers ont déjà été trouvés...
- Tester votre fonction en générant les nombres premiers jusqu'à 100, avec tous les affichages intermédiaires.
- Désactiver maintenant les affichages intermédiaires et tester avec $lim = 20000$. Que remarque-t-on ?

1.4 Comparaison des temps d'exécution

Pour confirmer l'impression d'efficacité de cet algorithme de crible, nous allons mesurer précisément le temps d'exécution.

- Importer le module `time`, qui fournit une fonction `time()` qui renvoie le temps système. Nous allons l'utiliser ici en stockant le temps avant appel d'une fonction dans une variable $t1$, le temps après appel dans une variable $t2$, et en affichant la différence entre ces 2 temps. Vérifiez que vous avez bien compris le fonctionnement de cette fonction.
- Écrire une fonction de test `comparer(lim)` qui reçoit une limite *lim*, utilise successivement les 2 fonctions `genererPremiers` et `criblerPremiers` pour générer la liste des nombres premiers jusqu'à *lim*, en mesurant leur temps d'exécution. Ne **pas** afficher la liste ainsi générée, mais afficher les deux temps d'exécution avec des messages clairs.

```
# par exemple
generation des nombres premiers jusqu'a 20000
basique : 1.9688389301300049 secondes
crible  : 0.04523801803588867 secondes
```

- Tester en appelant votre fonction `comparer` pour $lim = 20000$, puis pour $lim = 100000$. Que remarque-t-on ?