Session d'exercices : Fonctions

Nombres premiers

Un nombre premier est un entier positif (>0) qui admet exactement deux diviseurs: 1 et lui-même. Ainsi, 1 n'est pas premier car il n'a qu'un seul diviseur entier positif. Zéro n'est pas premier non plus car il est divisible par tous les entiers positifs. Mais, 2, 3, 5, 7, 11, 13, 17, 19,..., sont tous premiers car leurs seuls diviseurs sont 1 et eux-mêmes. Au contraire, par exemple, 8 n'est pas premier (on dit qu'il est *composé*), car $8 = 2 \times 4$ et $8 = 1 \times 8$.

Dans cet exercice, il faut écrire un programme appelé `nprimes.py` qui calcule les `n` premiers nombres premiers, les stocke dans un tableau (de taille `n`) et les affiche.

Pour ce faire, écrivez les définitions des fonctions suivantes:

a) `is_prime(p)` [Difficulté: **].

   Cette fonction prend un entier positif comme argument et renvoi `True` si ce nombre est premier ou `False` s'il ne l'est pas.

b) `find_prime_numbers(n)` [Difficulté: ***].

   Cette fonction prend un entier positif `n` comme argument et retourne un tableau contenant les `n` premiers nombres premiers.

Une fois ces fonctions complétées, terminez votre script pour qu'il demande à l'utilisateur d'entrer un nombre positif `n` puis qu'il affiche les `n` premiers nombres premiers.

Exemples de déroulement:

```
Entrez un nombre positif : 4
Les 4 premiers nombres premiers sont [2, 3, 5, 7].
```

```
Entrez un nombre positif : 10
Les 10 premiers nombres premiers sont [2, 3, 5, 7, 11, 13, 17, 19, 23, 29].
```

```
Entrez un nombre positif : -42
Erreur : votre nombre doit etre positif.
```

Nombres copremiers

En mathématiques, on dit de manières équivalentes:

- que deux entiers `a` et `b` sont premiers entre eux,
- que `a` est premier avec `b`,
- que `a` et `b` sont copremiers,
- ou que `a` et `b` sont étrangers

s'ils n'ont aucun facteur premier en commun.

En d'autres termes, `a` et `b` sont premiers entre eux s'ils n'ont aucun diviseur autre que 1 et -1 en commun. De manière équivalente, ils sont premiers entre eux si et seulement si leur plus grand diviseur commun est égal à 1.

Par exemple, 6 et 35 sont premiers entre eux, mais 6 et 27 ne le sont pas parce qu'ils sont tous les deux divisibles par 3. 1 est premier avec tout entier.

Voici un algorithme naïf et pas vraiment optimal qui détermine si les entiers `a` et `b`, plus grands que 1, sont premiers entre eux :

```
fonction COPRIME(a, b)
    c ← vrai                                  # true if a common divisor has been found
    pour i allant de 2 à min(a, b)
        si i est un diviseur de a et de b alors
            c ← faux
    retourner c
```

Lisez et analisez cet algorithme. Qu'affiche-t-il si `a = 6` et `b = 35` ? Et si `a = 6` et `b = 27` ?

En utilisant l'algorithme ci-dessus, érivez un script qui demande à l'utilisateur d'entrer deux nombres entiers supérieurs à 1 et qui affiche si il sont premiers entre eux ou pas.

Réalisez deux versions différentes selon les instructions suivantes:

a) [Difficulté: **] Créez un fichier `coprime_slow.py`. Implémenter une fonction `coprime` en utilisant l'algorithme exactement comme il est présenté ci-dessus, puis appelez cette fonction.

b) [Difficulté: *] Créez un fichier `coprime.py`. Cette fois, dans la fonction `coprime`, exécutez l'instruction `return false` dès que le premier diviseur commun est trouvé. Si la boucle se termine sans trouver de diviseur commun, la fonction doit renvoyer `True` à la fin. Dans ce cas, la variable c n'est pas nécessaire.

c) [Difficulté: **] Affichez les temps d'exécution de ces deux programmes pour `a = 7` et `b = 15`, ainsi que pour `a = 199 999 995` et `b = 200 000 000` (entrez ces nombres sans espaces).

   Une manière de mesurer le temps d'exécution d'un bout de code Python et d'enregistrer «l'heure» (à une très haute précision) avant et après son exécution grâce à la fonction `time.time()` du package `time`, puis d'afficher la différence entre les deux.

Pour mesurer le temps d'exécution de votre fonction `coprime`, vous pouvez donc utiliser le code suivant:

```
import time
start = time.time()
are_coprime = coprime(a, b)
end = time.time()
print(f"Calcule en {end - start:.5f} secondes.")
```

Essayez de comprendre pourquoi le temps d'exécution entre les deux versions de votre programme change.

Exemples de déroulement (version lente):

```
Entrez un premier nombre : 199999995
Puis un deuxieme : 200000000
Vos nombres ne sont pas premiers entre eux.
Calcule en 5.87773 secondes.
```

```
Entrez un premier nombre : 777777
Puis un deuxieme : 155555
Vos nombres sont premiers entre eux.
Calcule en 0.005 secondes.
```

```
Entrez un premier nombre : 10
Puis un deuxieme : 0
Erreur : vos nombres doivent etre positifs.
```