Épreuve d'informatique de l'X - 2009 - PSI/PT

(PSI*)

Question 1 : Si n admet un diviseur dans l'intervalle entier $[\![2, n-1]\!]$, c'est que n n'est pas un nombre premier (détecté en ligne 5) et donc le résultat sera `False`. Si n est premier, la variable `bool` n'est pas modifiée dans la boucle et la valeur retournée sera `True`.

```
1  def estPremier(n):
2      bool = True
3      for i in range(2,n):
4          if mod(n, i) == 0:
5              bool = False
6      if bool:
7          return(1)
8      else:
9          return(0)
```

Question 2 : On parcourt tous les entiers de 2 à n (boucle des lignes 4 à 7). À chaque fois que l'on trouve un nombre premier, on incrémente le compteur de nombres premiers (ligne 6) et on le stocke dans le tableau des nombres premiers (ligne 7). Le tableau `premier` est supposé être indexé à partir de 1. La variable `nbPremiers` permet de compter le nombre de nombres premiers.

```
1  def petitsPremiers(n):
2      global premier
3      nbPremiers = 0
4      for i in range(2,n+1):
5          if estPremier(i) == 1 :
6              nbPremiers = nbPremiers+1;
7              premier[nbPremiers] = i
8      return(nbPremiers);
```

Question 3 : On stocke 2 dans le tableau des nombres premiers (ligne 3) puis on examine les entiers de deux en deux (ligne 18) à partir de 3 (boucle des lignes 6 à 18). Pour un entier i donné, on teste la divisibilité de i avec les entiers premiers p ($p \geqslant 3$) déjà trouvés et qui vérifient $p^2 \leqslant i$ (boucle conditionnelle des lignes 10 à 14). Si on n'a trouvé aucun diviseur pour i, on incrémente le compteur de nombres premiers et on ajoute i au tableau des nombres premiers (lignes 15 à 17).

```
1  def petitsPremier2(n):
2      global premier
3      premier[1] = 2
4      nbPremiers = 1
5      i = 3
6      while i <= n:
7          j = 2
8          bool = True
9          p = 3
10         while p*p <= i and bool :
11             if mod(i,p) == 0:
12                 bool = False
13             j = j+1;
14             p = premier[j];
15         if bool :
16             nbPremiers = nbPremiers + 1
17             premier[nbPremiers] = i
18         i = i+2;
19     return(nbPremiers);
```

Question 4 : On écrit la fonction `petitsPremiers3` qui recherche les nombres premiers en suivant la méthode du crible d'Ératosthène. L'étape 1 correspond aux lignes 4 et 5. L'étape 2 correspond à la boucle conditionnelle des lignes 10 et 11. L'étape 3 correspond aux lignes 16 à 20. Quand on n'a coché aucune case, on quitte la boucle puis on compte et on place les nombres premiers trouvés dans le tableau `premier` (boucle des lignes 23 à 27).

```
1  def petitsPremiers3(n):
2      global premier;
3      raye = True;
4      tab = np.zeros(n)
5      caillou1 = 0
6      while raye :
7          raye = False;
8          caillou1 = caillou1 + 1
9          # on positionne la première pierre (étape 2).
10         while tab[caillou1] <> 0 :
11             caillou1 = caillou1+1
12         pas = caillou1 + 1
13         # on positionne la deuxième pierre.
14         caillou2 = caillou1 + pas;
15         # on biffe les cases (étape 3).
16         while caillou2 <=  n-1 :
17             if tab[caillou2] == 0:
18                 tab[caillou2] = 1
19                 raye = True
20             caillou2 = caillou2+pas
21     nbPremiers = 0
22     # on récupère les nombres premiers
23     for i in range(1,n):
24         if tab[i] <> 1 :
25             nbPremiers = nbPremiers+1
26             premier[nbPremiers] = i+1
27     return(nbPremiers)
```

Question 5 : L'étape 1, qui est une étape d'initialisation, est effectuée en lignes 3 et 4. L'étape 2 de l'algorithme de l'énoncé correspond à la boucle conditionnelle des lignes 6 à 11. On sort de cette boucle dès que m vaut 1. Les divisions successives ont lieu dans la boucle des lignes 7 à 10. Cette boucle correspond à l'étape 3. L'étape 4 correspond à la ligne 11.

```
1  def factoriser(n):
2      global premier, facteur;
3      m = n; i = 1
4      petitsPremiers3(n)
5      nbDiviseurs = 0
6      while m <> 1:
7          while mod(m, premier[i]) == 0 :
8              nbDiviseurs = nbDiviseurs + 1
9              facteur[nbDiviseurs] = premier[i]
10             m = m/premier[i]
11         i = i+1;
12     return(nbDiviseurs)
```

Question 6 : Dans cette fonction de factorisation, on n'utilise plus le tableau des nombres premiers. Le test des lignes 14 à 16 correspond à la seconde remarque de l'énoncé. Si le dernier m calculé est différent de 1, c'est que l'entier m un facteur premier de la décomposition de n, on le rajoute alors aux facteurs de n.

```
1  def factoriser2(n):
2      global facteur
3      m = n
4      i = 2
5      j = 1
6      nbDiviseurs = 0
7      while m <> 1 and i*i <= m:
8          while mod(m,i) <> 0 :
9              j = j+2
10             i = j
11         nbDiviseurs = nbDiviseurs + 1
12     facteur[nbDiviseurs] = i
13     m = m/i
14     if m <> 1 :
15         nbDiviseurs = nbDiviseurs + 1
16         facteur[nbDiviseurs] = m
17     return(nbDiviseurs)
```

Question 7 : Avec la fonction `factoriser`, on a rangé les facteurs premiers de n en ordre croissant dans le tableau `facteur` (ligne 3). On exploite ce tableau pour remplir le tableau `alpha`. C'est l'objet de la boucle des lignes 7 à 14. On traite le cas du dernier facteur premier de n aux lignes 15 et 16.

```
1  def calculerAlpha(n):
2      global facteur, alpha;
3      m = factoriser(n)
4      p = facteur[1]
5      nbFacteurs = 0
6      compt = 0
7      for i in range(1,m+1):
8          if facteur[i] == p:
9              compt = compt+1
10         else :
11             nbFacteurs = nbFacteurs+1
12             alpha[nbFacteurs] = compt
13             compt = 1
14             p = facteur[i]
15     nbFacteurs = nbFacteurs+1
16     alpha[nbFacteurs] = compt
17     return(nbFacteurs);
```

Question 8 : Pour tester si n est de la forme a^b, on commence par calculer les puissances des facteurs premiers de la décomposition de n (ligne 3), puis on vérifie que tous les ordres de multiplicité des facteurs premiers de la décomposition de n sont divisibles par b (boucle des lignes 5 à 6).

```
1  def estPuissance(n,b):
2      global alpha
3      m = calculerAlpha(n)
4      bool = True
5      for i in range(1, m+1):
6          bool = bool and (mod(alpha[i],b) == 0)
7      if bool :
8          return(1)
9      else :
10         return(0)
```

Question 9 : Pour détecter si n est un carré ou pas, on peut appliquer une méthode de Newton modifiée pour rechercher une racine de l'équation $x^2 = n$. La méthode de Newton converge très rapidement. Ici on effectue des divisions entières au lieu d'effectuer des divisions réelles. On arrête la boucle dès que la distance entre deux termes consécutifs de la suite est inférieure ou égale à 1. Il ne reste plus cas vérifier si la dernière valeur calculer est la racine carrée de n (ligne 8).

```
1  def estCarre(n) :
2      x = n
3      y = (x*x+n)//(2*x)
4      while abs(y - x) > 1 :
5          print(x)
6          x = y
7          y = (x*x+n)//(2*x)
8      return(y*y==n);
```

Ou bien, on peut tester si n est égal au carré de sa racine carrée entière :

```
1  def estCarre(n):
2      rac = int(np.sqrt(n))
3      return(n == rac*rac)
```

Question 10 : On applique directement la définition donnée par l'énoncé des nombres de Carmichael.

- on vérifie que c est impair non premier (ligne 3),
- on vérifie que les ordres de multiplicité des facteurs premiers de la décomposition de c sont égaux à 1 (ligne 6),
- on vérifie que les $p_i - 1$ divise $c - 1$ pour tous les facteurs premiers p_i entrant dans la décomposition de c (ligne 7).

```
1  def estCarmichael(c):
2      global facteur, alpha;
3      bool = ((c % 2 == 1) and (not(estPremier(c))))
4      m = calculerAlpha(c)
5      for i in range(1, m+1):
6          bool = bool and ((alpha[i] == 1)
7                  and ((c-1) % (facteur[i]-1) == 0))
8      if bool:
9          return(1)
10     else:
11         return(0)
```

Question 11 : On suit les consignes de l'énoncé. On calcule les nombres premiers de 3 à n (ligne 3). On calcule tous les produits de trois de ces nombres et on teste si ce sont des nombres de Carmichael (boucle des lignes 5 à 16). Si on a bien un nombre de Carmichael, on le stocke dans le tableau `carmichael` et on incrémente le compteur de nombres de Carmichael (ligne 12 à 16).

```
1  def calculerCarmichael3(n):
2      global premier, carmichael;
3      nbPrem = petitsPremiers3(n)
4      nbCarmichael = 0
5      for i in range(1, nbPrem-1):
6          for j in range(i+1,nbPrem):
7              for k in range(j+1,nbPrem+1):
8                  p1 = premier[i]
9                  p2 = premier[j]
10                 p3 = premier[k]
11                 c = p1*p2*p3
12                 if (c <= n) and ((c-1) % (p1-1)== 0)
13                             and ((c-1) % (p2-1)== 0)
14                             and ((c-1) % (p3-1)== 0):
15                     carmichael[nbCarmichael] = c
16                     nbCarmichael = nbCarmichael + 1
17     return(nbCarmichael)
```

Question 12 : On applique la méthode de criblage donnée par l'énoncé. On crée le tableau `t` (ligne 5) que l'on initialise (boucle des ligne 6 à 7). Dans la boucle des lignes 10 à 17, on effectue la division de `t`$[i]$ par p pour tous les entiers i de la forme $p^2 + k.p(p-1)$ plus petits que n. Dans la boucle des lignes 20 à 23, on récupère et on compte tous les nombres de Carmichael compris entre 1 et n.

```
1  def calculerCarmichael(n):
2      global premier, carmichael
3      nbPrem = petitsPremiers3(sqrt(n))
4      nbCarmichael = 0
5      t = np.zeros(n+1)
6      for i in range(n+1):
7          t[i] = i
8      i = 2
9      p = premier[i]; # on commence avec p=3
10     while p*p < n :
11         j = p*p
12         pas = p*(p-1)
13         while j <= n :
14             t[j] = t[j]//p
15             j = j+pas
16         i = i+1
17         p = premier[i]
18     nbCarmichael = 0
19     # on récupère les nombres de Carmichael
20     for i in range(2,n) :
21         if t[i] == 1 :
22             nbCarmichael = nbCarmichael+1
23             carmichael[nbCarmichael] = i
24     return(nbCarmichael)
```

Voici les nombres de Carmichael inférieurs ou égaux à 5000000 : 561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, 41041, 46657, 52633, 62745, 63973, 75361, 101101, 115921, 126217, 162401, 172081, 188461, 252601, 278545, 294409, 314821, 334153, 340561, 399001, 410041, 449065, 488881, 512461, 530881, 552721, 656601, 658801, 670033, 748657, 825265, 838201, 852841, 997633, 1024651, 1033669, 1050985, 1082809, 1152271, 1193221, 1461241, 1569457, 1615681, 1773289, 1857241, 1909001, 2100901, 2113921, 2433601, 2455921, 2508013, 2531845, 2628073, 2704801, 3057601, 3146221, 3224065, 3581761, 3664585, 3828001, 4335241, 4463641, 4767841, 4903921.