DS 1 d'informatique de l'année 2023/2024 - MPSI - Nombres premiers - Correction

Question 1.a –

```
# Solution 1
def listeDiv(n):
    assert n > 0
    L = []
    for d in range(1, n+1):
        if n % d == 0:
            L.append(d)
    return L
```

```
# Solution 2
def listeDiv(n):
    assert n > 0
    return [d for d in range(1, n+1) if n % d == 0]
```

Question 1.b –

```
# Suppose n >= 1
def estPremier1(n):
    return len(listeDiv(n)) == 2
```

Question 2 –

```
def listeNP():
    return [n for n in range(1,4568) if estPremier1(n)]
```

Question 3 –

```
def estPremier2(n):
    if n == 2:
        return True
    if n == 1 or n % 2 == 0:
        return False
    for d in range(3, math.floor(n**0.5)+1, 2):
        if n % d == 0:
            return False
    return True
```

Question 4.a –

```
# Solution 1
def makeP(n):
    P = []
    for a in range(1, n):
        if math.gcd(n, a) == 1:
            P.append(a)
    return P
```

```
# Solution 2
def makeP(n):
    return [a for a in range(1, n) if math.gcd(n, a) == 1]
```

Question 4.b –

```
# On suppose que n n'est pas premier
def estCarmic(n):
    if n == 1:
        return False
    for a in makeP(n):
        if (a**(n-1) - 1) % n != 0:
            return False
    return True
```

Question 5.a –

```
def listeBoolNP():
    L = [False]*4568
    for p in listeNP():
        L[p] = True
    return L
```

Question 5.b –

```
# Solution 1
def indicesFalse(L):
    M = []
    for i in range(2, len(L)):
        if not L[i]:
            M.append(i)
    return M
```

```
# Solution 2
def indicesFalse(L):
    return [i for i in range(2, len(L)) if not L[i]]
```

Question 5.c –

```
# Solution 1
def listeCarmic():
    M = indicesFalse(listeBoolNP())
    NC = []
    for n in M:
        if estCarmic(n):
            NC.append(n)
    return NC
```

```
# Solution 2
def listeCarmic():
    M = indicesFalse(listeBoolNP())
    return [n for n in M if estCarmic(n)]
```

Question 6 –

```
def val2Adique(m):
    for k in range(1, m+1):
        if m % 2**k != 0:
            return k-1
    return None # Cette ligne n'est jamais exécutée
```

Question 7.a –

```
def verifC(n, a):
    return n >= 4 and n % 2 == 1 and a >= 2 and a <= n-2
```

Question 7.b –

```
def get_u(n, a):
    v2 = val2Adique(n-1)
    d = (n-1)//2**v2
    L = [pow(a, d, n)]
    for _ in range(v2):
        L.append(L[-1]**2 % n)
    return L
```

Question 7.c – La fonction `verifC` a pour signature :

```
verifC(n: int, a: int) -> bool
```

La fonction `get_u` a pour signature :

```
get_u(n: int, a: int) -> list[int]
```

Question 8 –

```
def estTemoinMiller(n, a):
    assert verifC(n, a)
    L = get_u(n, a)
    if L[0] == 1 or L[0] == n-1:
        return False
    if L[-1] != 1:
        return True
    for k in range(1, len(L)):
        if L[k-1] != n-1 and L[k-1] != 1 and L[k] == 1:
            return True
    return False
```

Question 9 –

```
def estPremier3(n):
    if n == 2 or n == 3:
        return True
    if n <= 1 or n % 2 == 0:
        return False
    for _ in range(100):
        a = random.randint(2, n-2)
        if estTemoinMiller(n, a):
            return False
    return True
```

Question 10 –

```
def tousPremiers(L):
    for n in L:
        if not estPremier3(n):
            return False
    return True
```

Question 11 –

```
# Solution 1
def repunitPremiers(k):
    L = []
    for i in range(1, k+1):
        n = int("1"*i)
        if estPremier3(n):
            L.append(n)
    return L
```

```
# Solution 2
def repunitPremiers(k):
    L = [int("1"*i) for i in range(1, k+1)]
    return [n for n in L if estPremier3(n)]
```

Question 12 –

```
def estNPPalind(n):
    if not estNPremier3(n):
        return False
    s = str(n)
    for i in range(len(s)//2):
        if s[i] != s[len(s)-1-i]:
            return False
    return True
```

Question 13 – On génère la liste de tous les entiers obtenus lorsqu'on applique une permutation circulaire.

```
# Solution 1
def estNPCirc(n):
    L = [n]
    s = str(n)
    for _ in range(len(s)-1):
        s = s[1:] + s[0]
        L.append(int(s))
    return tousPremiers(L)
```

```
# Solution 2
def estNPCirc(n):
    s = str(n)
    L = [int(s[i:] + s[:i]) for i in range(len(s))]
    return tousPremiers(L)
```

Question 14 – On génère la liste de tous les entiers obtenus lorsqu'on supprime des chiffres à droite de n.

```
# Solution 1
def estNPTD(n):
    T = [n]
    s = str(n)
    for _ in range(len(s)-1):
        s = s[:-1]
        T.append(int(s))
    return tousPremiers(T)
```

```
# Solution 2
def estNPTD(n):
    s = str(n)
    T = [int(s[:i]) for i in range(1, len(s)+1)]
    return tousPremiers(T)
```

Question 15 – Soit $k \in \mathbb{N}^*$ et n un élément de L_{k+1}. Alors n est un nombre premier tronquable à droite composé de $k+1$ chiffres (avec $k+1 \geqslant 2$).

On note m l'entier obtenu lorsqu'on supprime le chiffre des unités de n. Alors, par définition, m est un nombre premier tronquable à droite. Ainsi, $m \in L_k$.

De plus, le chiffre des unités de n est nécessairement 1, 3, 7 ou 9, sinon n ne serait pas premier car divisible par 2 ou 5. Finalement, on a bien $n \in M_k$.

Question 16.a –

```
def L_to_M(L):
    M = []
    for n in L:
        M.append(n*10 + 1)
        M.append(n*10 + 3)
        M.append(n*10 + 7)
        M.append(n*10 + 9)
    return M
```

Question 16.b –

```
def M_to_L(M):
    return [n for n in M if estPremier3(n)]
```

Question 16.c –

```
def make_T8_T9():
    L = [2, 3, 5, 7]
    T = L[:]
    for _ in range(7):
        L = M_to_L(L_to_M(L))
        T = T + L
    L9 = M_to_L(L_to_M(L))
    T9 = T + L9
    return T, T9
```

Question 17 – D'après le programme donné dans l'énoncé, on a $T_8 = T_9$. En d'autres termes, il n'y pas de nombres premiers tronquables à droite avec exactement 9 chiffres. Ainsi, il n'existe pas de nombres premiers tronquables à droite avec plus de 9 chiffres. En conclusion :

Il y a exactement 83 nombres premiers tronquables à droite.

Question 18 – On définit les listes L_k, M_k et T_k comme dans le cas des nombres premiers tronquables à droite. La différence est que la liste M_k est obtenue en concaténant les chiffres 1,2,3,4,5,6,7,8 ou 9 au début d'un élément de L_k.

Si on admet que le plus grand nombre premier tronquable à gauche possède 24 chiffres (voir l'énoncé), il suffit de calculer T_{24} pour tous les obtenir.

```
# Renvoie une liste contenant les 9 nombres obtenus lorsqu'on ajoute un chiffre
# à gauche de n
def ajout_chiffre(n):
    """ajout_chiffre(n: int) -> list[int]"""
    L = []
    for c in ['1', '2', '3', '4', '5', '6', '7', '8', '9']:
        L.append(int(c + str(n)))
    return L
```

```
def L_to_M_bis(L):
    """L_to_M_bis(L: list[int]) -> list[int]"""
    M = []
    for n in L:
        M = M + ajout_chiffre(n)
    return M
```

```
def listeNPTG():
    L = [2, 3, 5, 7]
    T = L[:]
    for _ in range(23):
        M = L_to_M_bis(L)
        L = M_to_L(M)
        T = T + L
    return T
```

Question 19 –

```
# Renvoie une liste contenant toutes les chaînes de caractères obtenues
# lorsqu'on remplace s[i] par un autre chiffre.
def modifChiffre(s, i):
    """modifChiffre(s: str, i: int) -> list[str]"""
    L = []
    for c in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']:
        if s[i] != c:
            L.append(s[:i] + c + s[i+1:])
    return L
```

```
# Renvoie la liste composée de tous les nombres obtenus lorsqu'on modifie un
# chiffre de n.
def modifDelicat(n):
    """modifDelicat(n: int) -> list[int]"""
    L = []
    s = str(n)
    for i in range(len(s)):
        L = L + modifChiffre(s, i)
    return [int(s) for s in L]
```

```
def estNPDelicat(n):
    if not estPremier3(n):
        return False
    L = modifDelicat(n)
    for k in L:
        if estPremier3(k):
            return False
    return True
```

Question 20 – Pour écrire ce programme, on va générer la liste de tous les nombres obtenus lorsqu'on permute les chiffres de `n`. Pour cela, on va utiliser des listes de type `list[str, str]` notées L_0, L_1, …, L_k où `k` est le nombre de chiffres dans `n`. Ces listes sont définies par récurrence :

- L_0 = `[(str(n), "")]`
- Pour chaque `i`, pour chaque couple `(s1, s2)` dans L_i, pour chaque caractère `c` de `s1`, soit s_1' la chaîne de caractères `s1` dans laquelle on a supprimé `c` et s_2' la chaîne de caractères `s2` à laquelle on a concaténé `c`. La liste L_{i+1} est alors définie comme tous les couples (s_1', s_2') qu'on peut obtenir à partir des couples (s_1, s_2) de L_i.

Par exemple, avec `n = 1379`, on a :

```
L0 =[("1379","")]
L1 =[("379","1"), ("179","3"), ("139","7"), ("137","9")]
L2 =[("79","13"), ("39","17"), ("37","19"),
       ("79","31"), ("19","37"), ("17","39"),
       ("39","71"), ("19","73"), ("13","79"),
       ("37","91"), ("17","93"), ("13","97")]
...
```

Avec cette définition, la dernière liste L_k contient tous les couples `("", s)` où `s` est une permutation des chiffres de `n`.

```python
# Calcule L_{i+1} à partir de L_{i}
def etapePerm(L):
    """etapePerm(L: list[str, str]) -> list[str, str]"""
    M = []
    for (s1, s2) in L:
        M = M + [(s1[:i]+s1[i+1:], s2+s1[i]) for i in range(len(s1))]
    return M
```

```python
# Renvoie la liste de toutes les permutations de s
def toutesPerm(s):
    L = [(s, "")]
    for _ in range(len(s)):
        L = etapePerm(L)
    return [s for _,s in L]
```

```python
def estNPPerm(n):
    L = toutesPerm(str(n))
    L = [int(s) for s in L]
    return tousPremiers(L)
```