Exercice 1. Fonction mystère
On considère la fonction suivante:

```python
def mystere(L, k):
    if k == len(L) - 1:
        return True
    return L[k] < L[k + 1] and mystere(L, k + 1)
```

1. Quelle valeur renvoie `mystere([1, 3, 4, 7], 0)`? `mystere([3, 1, 2], 0)`?
   ▶ `mystere([1, 3, 4, 7], 0)` renvoie `True`. `mystere([3, 1, 2], 0)` renvoie `False`.

2. Expliquer à quoi sert cette fonction `mystere`.
   ▶ Elle détermine si une liste `L` est triée par ordre croissant, à partir de l'indice `k`.

Exercice 2. Somme des chiffres
Écrire une fonction `somme` récursive renvoyant la somme des chiffres d'un entier. Par exemple, `somme(483)` doit renvoyer 15 $(= 4 + 8 + 3)$. Il est interdit d'utiliser une boucle `while` ou `for`. On pourra utiliser `//` et `%`.
▶ On obtient le chiffre des unités de n avec $n\%10$. À ce nombre il faut rajouter la somme des autres chiffres, obtenue avec `somme(n // 10)`.

```python
def somme2(n):
    if n == 0: return 0
    return (n % 10) + somme2(n // 10)
```

Exercice 3. Méthode de Newton récursive
La méthode de Newton pour approximer une solution de $f(x) = 0$ consiste à définir une suite u_n définie de la façon suivante (en supposant que tout est bien défini):

- u_0 est quelconque (mais de préférence proche du zéro qu'on veut approximer)
- Pour tout $n \geq 0$, $u_{n+1} = u_n - \dfrac{f(u_n)}{f'(u_n)}$

Écrire une fonction récursive `newton(f, fp, u0, n)` renvoyant le $n$ème terme u_n de la suite définie ci-dessus, avec `fp` la dérivée de `f` et $u_0 =$ `u0`.
▶ 1ère solution (récupérer u_{n-1} puis renvoyer u_n):

```python
def newton(f, fp, u0, n):
    if n == 0: return u0
    u = newton(f, fp, u0, n - 1)
    return u - f(u)/fp(u)
```

2ème solution (modifier u_0):

```python
def newton(f, fp, u0, n):
    if n == 0: return u0
    return newton(f, fp, u0 - f(u0)/fp(u0), n - 1)
```

Exercice 4. E3A 2017
1. Soient a et b deux réels, $f : [a, b] \to \mathbb{R}$ une fonction continue telle que $f(a)f(b) < 0$.
   (a) Justifier que f s'annule sur $[a, b]$.
   (b) Écrire une fonction Python `rech_dicho` prenant en arguments une fonction `f`, deux flottants `a` et `b` tels que `f(a)f(b)<0` et une précision `eps` et qui renvoie un couple de réels encadrant un zéro de `f` à une précision `eps` près. rech_dicho devra être récursif.
2. Soit f une fonction continue de $[0, 1]$ dans $[0, 1]$.
   (a) Montrer que f admet un point fixe (c'est-à-dire un réel c de $[0, 1]$ tel que $f(c) = c$).
   (b) Écrire une fonction Python `rech_pt_fixe` qui prend en argument une fonction `f` que l'on suppose continue de $[0, 1]$ dans $[0, 1]$, un précision `eps` et qui renvoie un couple de réels encadrant un point fixe de `f` à une précision `eps` près. On pourra utiliser la fonction `rech_dicho`.

▶ Corrigé :

1.a $f(a)$ et $f(b)$ sont de signes opposés et f est continue donc s'annule sur $[a, b]$ d'après le théorème des valeurs intermédiaires.

1.b

```python
def rech_dicho(f, a, b, eps):
    if b - a < eps: return (a, b)
    m = (a + b)/2
    if f(a)*f(m) < 0: # f s'annule sur [a, m]
        return rech_dicho(f, a, m, eps)
    else:
        return rech_dicho(f, m, b, eps)
```

2.a Soit $g : x \longmapsto f(x) - x$. g est continue, $g(0) = f(0) \geq 0$ et $g(1) = f(1) - 1 \leq 1$ donc, d'après le théorème des valeurs intermédiaires, il existe $c \in [0, 1]$ tel que $g(c) = 0$. On a alors $f(c) = c$.

2.b Petite difficulté: définir une fonction à l'intérieur d'une autre fonction.

```python
def rech_pt_fixe(f, a, b, eps):
    def g(x):
        return f(x) - x
    return rech_dicho(g, a, b, eps)
```

Exercice 5. E3A 2016

On considère la fonction définie comme suit en python :

```python
def M(n) :
    if n > 100 :
        return n - 10
    else :
        return M (M (n + 11))
```

2.1 Que fait l'appel `M(101)` ?

2.2 Plus généralement, que fait l'appel `M(N)` si $\texttt{N} > 100$?

2.3 Que renvoient l'appel `M(100)` ? Puis `M(99)`, `M(98)` ?

2.4 Conjecturer ce que renvoie l'appel `M(N)` où $\texttt{N} \leqslant 100$, entier naturel, puis le démontrer.

▶ Corrigé :

1. Dans l'appel `M(101)` on renvoie directement 91.

2. Si $\texttt{N} > 100$ on est de même dans un cas de base de la fonction et on renvoie $\texttt{N} - 10$.

3. Dans l'appel `M(100)`, on calcule `M(111)` qui vaut 101 avec lequel on rappelle `M` pour obtenir 91. Dans l'appel `M(99)`, on calcule `M(110)` qui vaut 100 avec lequel on rappelle `M` pour obtenir 91. Dans l'appel `M(98)`, on calcule `M(109)` qui vaut 99 avec lequel on rappelle `M` pour obtenir 91.

4. On peut penser que pour tout $\texttt{N} \leq 100$ on a `M(N)` qui vaut 91. On le prouve par récurrence descendante sur `N`.

   - <u>Initialisation</u> : c'est vrai si $\texttt{N} = 100$.
   - <u>Hérédité</u> soit $\texttt{N} \leq 100$ tel que le résultat soit vrai du rang 100 jusqu'au au rang `N`. L'appel `M(N-1)` déclenche celui de `M(N+10)`. Si $\texttt{N} + 10 > 100$ cet appel renvoie `N` et par hypothèse de récurrence, le second appel à `M` donne 91. Sinon $\texttt{N} + 10$ est entre `N` et 100 et par hypothèse de récurrence l'appel donne 91 et le second appel à `M` (avec 91) donne 91.

Pour la dernière question, on peut aussi prouver, par récurrence classique: $\forall n \in \{0, 1, ..., 100\}$, `M(100-n)` renvoie 91.