Récursion

Les exercices suivants doivent tous être résolus à l'aide d'une fonction récursive, les boucles sont interdites.

Exercice 1.1 (`fact`)

1. Ci-dessous, deux définitions de la fonction factorielle.

$$fact(n) = \begin{cases} 1 & \text{si } n = 0, \\ n * fact(n-1) & \text{si } n \geq 1. \end{cases}$$

$$fact(n) = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$$

Ci-dessous sont proposées plusieurs versions d'une fonction calculant la factorielle d'un entier $n \geq 0$. Pour chacune de ces versions, indiquer la valeur de la variable `OUTPUT` après l'exécution du programme. Si une erreur se produit, préciser sa nature.

(a)
```python
def fact(n):
    if n < 2:
        return 1
    else:
        fact(n - 1)
OUTPUT = fact(4)
```

(b)
```python
def rec_fact(n):
    if n < 2:
        return res
    else:
        res = res * rec_fact(n - 1)
def fact(n):
    res = 1
    rec_fact(n)
OUTPUT = fact(4)
```

(c)
```python
def rec_fact(n, res):
    if n < 2:
        return res
    else:
        res = res * rec_fact(n - 1, res)
def fact(n):
    res = 1
    rec_fact(n, res)
OUTPUT = fact(4)
```

(d)
```python
def rec_fact(n, res):
    if n < 2:
        return res
    else:
        res = res * rec_fact(n - 1, res)
        return res
def fact(n):
    res = 1
    return rec_fact(n, res)
OUTPUT = fact(4)
```

2. Ecrire une fonction calculant la factorielle d'un entier $n \geq 0$.

Exercice 1.2 (`fibo_calls`)

La suite de Fibonacci est définie par :

$$fibo(n) = \begin{cases} 1 & \text{si } n \leq 1, \\ fibo(n-1) + fibo(n-2) & \text{sinon.} \end{cases}$$

1. On considère le code suivant, censé calculer récursivement un terme de la suite de Fibonacci.
   Ce code est incorrect. Identifier l'erreur et expliquer pourquoi le nombre d'appels récursifs n'est pas correctement compté.

```python
def fibo_rec(n, nb_calls):
    nb_calls += 1
    if n < 2:
        return 1
    else:
        return fibo_rec(n-1, nb_calls) + fibo_rec(n-2, nb_calls)

def fibo_calls_wrong(n):
    nb_calls = 0
    res = fibo_rec(n, nb_calls)
    return (res, nb_calls)
```

2. Écrire la fonction `fibo_calls(n)` qui retourne un couple `(value, nb_calls)` tel que :
   — `value` est la valeur du n-ième terme de la suite de Fibonacci ;
   — `nb_calls` est le nombre total d'appels récursifs effectués.
   Exemples d'applications :

```
    >>> fibo_calls(0)
    (1, 1)
    >>> fibo_calls(1)
    (1, 1)
    >>> fibo_calls(4)
    (3, 9)
```

Exercice 1.3 (`binary_search`)

On considère le code suivant, censé implémenter une recherche dichotomique récursive dans une liste triée.
Ce code est incorrect. Identifier l'erreur et expliquer pourquoi l'algorithme ne fonctionne pas correctement, puis corriger la fonction.

```python
def __binary_search_wrong(L, x, left, right):
    if left > right:
        return -1
    else:
        mid = left + (right - left) // 2
        if x == L[mid]:
            return mid
        else:
            if x < L[mid]:
                __binary_search_wrong(L, x, left, mid)
            else:
                __binary_search_wrong(L, x, mid+1, right)

def binary_search_wrong(L, x):
    return __binary_search_wrong(L, x, 0, len(L))
```

Exercice 1.4 (`reverse`)

Écrire la fonction `reverse(n)` qui retourne le nombre formé par les chiffres de n dans l'ordre inverse.
Exemples d'applications :

```
>>> reverse(1234)
4321
>>> reverse(5089)
9805
>>> reverse(-5089)
-9805
>>> reverse(1)
1
>>> reverse(0)
0
```

Exercice 1.5 (`nth_prime`)

1. Écrire la fonction `is_prime(k)` qui vérifie si un entier $k > 1$ est premier.
   Exemples d'applications :

```
>>> is_prime(2)
True
>>> is_prime(15)
False
>>> is_prime(17)
True
```

2. Écrire la fonction `nth_prime(n)` qui retourne le n-ième nombre premier ($n > 1$). Vous devez impérativement utiliser la fonction précédente `is_prime`.
   Exemples d'applications :

```
>>> nth_prime(1)
2
>>> nth_prime(5)
11
>>> nth_prime(10)
29
```

Exercice 1.6 (`binary_search_calls`)

Écrire la fonction `binary_search_calls(L, x)` qui effectue une recherche dichotomique de x dans la liste triée L et retourne un couple `(nb_calls, pos)` tel que :
- `nb_calls` est le nombre d'appels récursifs effectués.
- `pos` est la position de x dans L, ou `-1` si x n'est pas présent

Exemples d'applications :

```
>>> binary_search_calls([1, 3, 5, 7, 9], 5)
(1, 2)
>>> binary_search_calls([1, 3, 5, 7, 9], 4)
(3, -1)
>>> binary_search_calls([], 3)
(1, -1)
```