Quizz_2023_06

April 26, 2023

Ecrire une fonction `fibo()` qui prend comme paramètre un entier n et crée une liste d'entiers de longueur $n+1$ comprenant dans l'ordre les nombres de la suite de Fibonacci jusqu'au rang n. La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent.

Le terme de rang n, noté f_n, est défini $f_0 = 0, f_1 = 1$, et $f_n = f_{n-1} + f_{n-2}$ pour $n \geqslant 2$.

Par exemple, l'appel `fibo(5)` créera une liste comprenant les nombres : 0 1 1 2 3 5

Première proposition: algorithme itératif (= non récursif).

On construit progressivement la liste de tous les termes de la suite. Dans le cas où n vaut 0, la liste à renvoyer comprend 1 seul élément, et comme la liste initiale est créée avec 2 éléments, on ne renvoie pas la liste en entier (`return fb`) mais la tranche de 0 à $n+1$. Il n'y a pas d'autre cas particulier à prendre en compte (si on fait l'hypothèse que n est positif ou nul).

[14]:
```
def fibo(n):
    fb = [0, 1]
    for i in range(2,n+1):
        fb.append(fb[i-1]+fb[i-2])
    return fb[0:n+1]
```

[15]:
```
print(fibo(0))
print(fibo(5))
print(fibo(19))
```

```
[0]
[0, 1, 1, 2, 3, 5]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181]
```

Deuxième proposition: avec deux variables `a` et `b`

Algorithme itératif.

Les variables `a` et `b` contiennent en permanence deux termes successifs de la suite, et on utilise l'affectation conjointe de python pour avoir une écriture plus compacte.

[24]:
```
def fibo_ab(n):
    fb = []
    a, b = 0, 1
```

```
    fb.append(a)
    for __ in range(n):
        a, b = b, a+b
        fb.append(a)
    return fb
```

[25]:
```
print(fibo_ab(19))
print(fibo_ab(5))
print(fibo_ab(1))
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181]
[0, 1, 1, 2, 3, 5]
[0, 1]
```

La version précédente est inspirée d'une version classique de l'algo de Fibonacci, qui renvoie la valeur du $(n+1)$-ième terme de la suite (ie du terme F_n), donnée ci-dessous (il faut calculer tous les termes précédents, mais seule la valeur demandée est renvoyée).

[26]:
```
def fib(n):
    a, b = 0, 1
    for __ in range(n):
        a, b = b, a+b
    return a
```

[27]:
```
print(fib(0))
print(fib(3))
print(fib(5))
print(fib(16))
```

```
0
2
5
987
```

Version un peu plus explicite, où les différents cas limites ($n = 0$, $n = 1$) sont pris en considération:

[28]:
```
def fibo_cp(n):
    fb = []
    a, b = 0, 1
    if n >= 0: fb.append(a)
    if n >= 1: fb.append(b)
    for i in range(1,n):
        new = a + b
        fb.append(new)
        a, b = b, new
    return fb
```

[30]:
```
for i in range(11):
    print(fibo_cp(i))
```

```
[0]
[0, 1]
[0, 1, 1]
[0, 1, 1, 2]
[0, 1, 1, 2, 3]
[0, 1, 1, 2, 3, 5]
[0, 1, 1, 2, 3, 5, 8]
[0, 1, 1, 2, 3, 5, 8, 13]
[0, 1, 1, 2, 3, 5, 8, 13, 21]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Troisième proposition: version récursive

Il est naturel, étant donnée la définition des termes de la suite, de définir le terme n en utilisant la fonction elle-même pour calculer les termes $n-1$ et $n-2$. Voici tout d'abord un version naïve de la fonction qui calcule la valeur du terme F_n.

[31]:
```
def fibr(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibr(n-1) + fibr(n-2)
```

Il peut être tentant d'utiliser cet algorithme pour répondre à l'exercice, avec un simple boucle:

[34]:
```
def list_fibr(n):
    fb = []
    for i in range(n+1):
        fb.append(fibr(i))
    return fb
```

[35]:
```
print(list_fibr(3))
print(list_fibr(5))
print(list_fibr(21))
```

```
[0, 1, 1, 2]
[0, 1, 1, 2, 3, 5]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181, 6765, 10946]
```

Cependant, en terme de complexité, c'est un choix particulièrement coûteux, parce qu'on refait inutilement des calculs qui ont déjà été faits, aussi bien dans la boucle `list_fibr()` que dans la fonction `fibr()` elle-même.