Quizz_2022_06

March 13, 2022

Ecrire une fonction `fibo()` qui prend comme paramètre un entier n et **affiche** 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.

Notée F_n, elle est définie par $F_0 = 0, F_1 = 1$, et $F_n = F_{n-1} + F_{n-2}$ pour $n \geqslant 2$.

Par exemple, l'appel `fibo(5)` affichera: `0 1 1 2 3`

Remarque: Les n premiers termes sont les termes de F_0 à F_{n-1}. Autrement dit, le terme F_n est le $(n+1)$-*ième* terme de la suite. La terminologie de l'énoncé était ambiguë car on peut comprendre l'expression *le rang n* comme faisant référence au *n-ième* terme, donc F_{n-1}, ou comme faisant référence à F_n, donc au $(n+1)$-*ième* terme.

Dans les algorithmes ci-dessous, on utilise n dans certains cas pour représenter le *nombre* de termes de la suite (ce qui veut dire qu'on s'arrête à F_n) et dans certains cas pour faire référence à l'indice de F. Attention à bien distinguer les deux cas.

Première proposition: construction de la liste des n premiers termes

Algorithme itératif (= non récursif).

On construit progressivement la liste de tous les termes de la suite. Une fois la liste construite, il ne reste plus qu'à l'imprimer. On peut le faire dans la même fonction (`fiblistp()`) (noter le petit truc: au lieu d'imprimer la liste complète, on imprime la liste de 0 à n: ça évite de faire des tests pour les cas particuliers où le paramètre n vaut 0 ou 1). On peut aussi estimer que la construction de la liste et l'impression devraient être faits séparément (`fiblist()`et `print_fiblist()`).

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

[2]:
```
fiblistp(0)
fiblistp(5)
fiblistp(21)
```

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

[3]:
```
def fiblist(n):
    fb = [0, 1]
    for i in range(2,n):
        fb.append(fb[i-1]+fb[i-2])
    return fb[0:n]

def print_fiblist(n):
    print(fiblist(n))
```

[4]:
```
print_fiblist(1)
print_fiblist(3)
print_fiblist(16)
```

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

Deuxième proposition: avec deux variables `a` et `b` pour les deux éléments précédents

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. La version ci-dessous marche pour les valeurs de $n \geq 2$, mais pour la valeur 1 il faut prendre des précautions supplémentaires (voir plus loin ci-dessous).

[5]:
```
def fibp(n):
    a, b = 0, 1
    print(a, end=' ')
    for __ in range(n-1):
        a, b = b, a+b
        print(a, end=' ')
    print()
```

[6]:
```
fibp(21)
fibp(5)
fibp(1)  # affichage erroné
```

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

La version précédente est inspirée d'une version classique de l'algo de Fibonacci, qui calcule la valeur du $(n+1)$-ième terme de la suite (ie du terme F_n), donnée ci-dessous.

[7]:
```
def fib(n):
    a, b = 0, 1
    for __ in range(n):
        a, b = b, a+b
    return a
```

[8]:
```
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:

[9]:
```
def fibo(n):
    a, b = 0, 1
    if n >= 0: print(a, end=' ')
    if n >= 1: print(b, end=' ')
    for i in range(1,n):
        new = a + b
        print(new, end =' ')
        a, b = b, new
```

[10]:
```
for i in range(10):
    fibo(i)
    print()
```

```
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
```

Troisième proposition: programmes récursifs.

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.

[11]:
```
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:

[12]:
```
def print_fibr(n):
    for i in range(n):
        print(fibr(i), end=' ')
    print()
```

[13]:
```
print_fibr(3)
print_fibr(5)
print_fibr(21)
```

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

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 `print_fibr()` que dans la fonction `fibr()` elle-même.

On peut trouver une méthode récursive et efficace pour calculer les termes de Fibonacci, en utilisant ce qu'on appelle la *récursivité terminale* (l'instruction `return` est la dernière instruction exécutée). Dans cette version, les paramètres `a` et `b` vont désigner les deux termes précédant le terme courant, on parle d'*accumulateurs* dans ce genre d'implémentation.

[14]:
```
def fibRt(n, a, b):
    if n == 0: return a
    if n == 1: return b
    return fibRt(n-1, b, a+b)
```

[15]:
```
print(fibRt(5,0,1))
```

```
5
```

La fonction récursive ayant deux arguments supplémentaires, il faut utiliser une fonction *wrapper*:

[16]:
```
def fibonacci_rt(n):
    return fibRt(n,0,1)
```

[17]:
```
print(fibonacci_rt(2))
print(fibonacci_rt(9))
```

```
1
34
```

La possibilité d'avoir des valeurs par défaut pour les paramètres en python permet de se passer de cette fonction *wrapper* :

[18]:
```
def fibonacciRt(n, a=0, b=1):
    if n == 0: return a
    if n == 1: return b
    return fibonacciRt(n-1, b, a+b)
```

[19]:
```
print(fibonacciRt(20))
```

```
6765
```

Pour répondre à l'exercice, il faut trouver un moyen d'afficher les nombres dans le bon ordre. Avec la façon dont notre récursion est calculée, c'est assez simple:

[20]:
```
def fibonacciRtP(n, a=0, b=1):
    print(a, end=' ')
    if n == 0: return a
    if n == 1: return b
    return fibonacciRtP(n-1, b, a+b)
```

[21]:
```
fibonacciRtP(21)
```

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

[21]:
```
10946
```