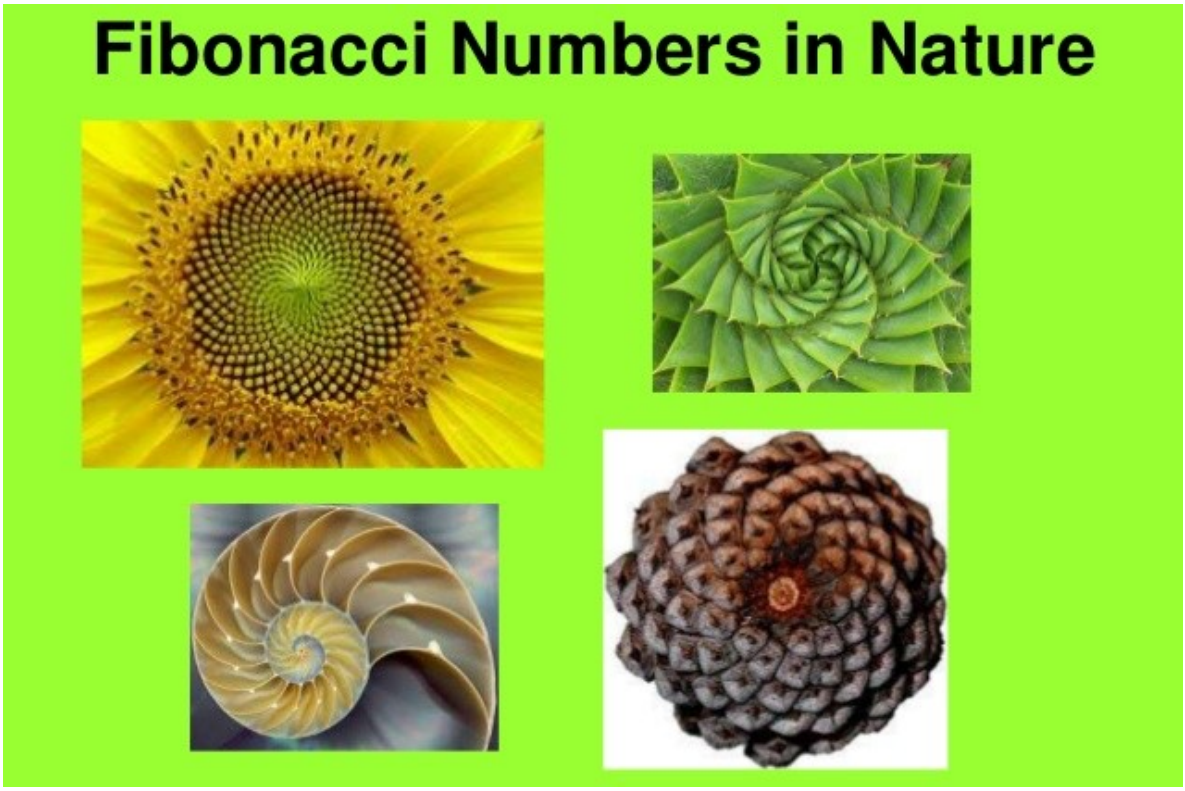
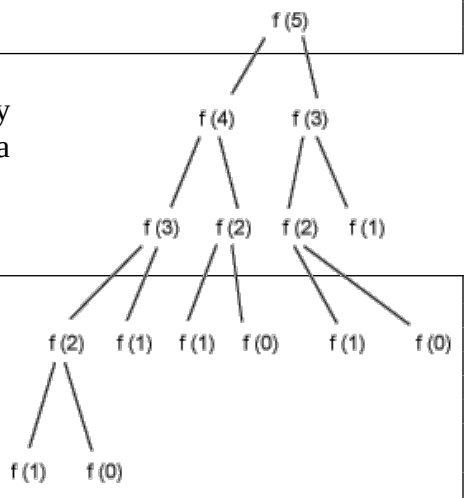



**Attention : Java Code**

Here is a RECURSIVE function to compute the Fibonacci sequence of number n. We can notice that the computing time is exponential in n : for instance we compute 3 times the function f(2) to compute f(5).

```
public int fibonacci_Recur(int n) {
   if (n == 0) {
      return 0;
   } else if (n == 1) {
      return 1;
   } else {
      return fibonacci_Recur(n - 1) + fibonacci_Recur(n - 2);
// Tree threads left and right
   }
}
```




A non recursive version hence ITERATIVE reduces complexity of computing. But we need more memory to store values in a table.

Notice that we start with f(0) and f(1) here.

```
public int fibonacci_Iter(int n) {
   int[] table = new int[n + 1];
   for (int i = 0; i < table.length; i++) {
      if (i == 0) {
         table[i] = 0;
      } else if (i == 1) {
         table[i] = 1;
      } else {
         table[i] = table[i - 2] + table[i - 1];
      }
   }
return table[n] ;
}
```

This is java code program it in Python and test it.

Then design your code into two functions FibRec(n) et FibIt(n).

In python, it codes :

- for the recursive way:

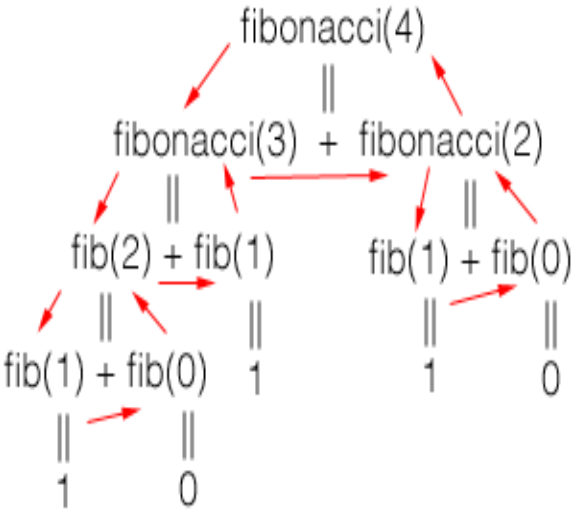
```
<<fibonacci_recursive.py>>=
def fibRec(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibRec(n-1) + fibRec(n-2)
```

```
Testing:
>>> fibRec(0), fibRec(1), fibRec(2), fibRec(3)
(0, 1, 1, 2)
>>> fibRec(7)
13
```

- For the iterative way :

```
#!/usr/bin/python
print "Liste des N premiers elements de la suite de Fibonacci"
def fibIt(n):
        a, b = 0, 1
        for i in range(n):
                print b,
                a, b = b, a+b
```

How to print something in the recursive function in order to get the feeling of the tree like recursion process ?




With the help of the *time* module and function *clock() for instance*, check out the complexity ratio in term of computing time (use n>>1 ).

Modify *FibIt* in order not only to display in the loop but also to return the *fibo(n)* value. (*return* in the def function and *print* in the main function).

Modify it again to generate the sequence in an infinite way.

```
def fibonacci(number, depth = 0):
    print " " * depth, number
    if number == 0:
        return 0
    elif number == 1:
        return 1
    else:
        return fibonacci(number-1, depth + 1) + fibonacci(number-2,
depth + 1)
```

```
<<fibonacci_iteration.py>>=
def fib(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a
```

```
<<fibonacci_generator.py>>=
def fib():
    a, b = 0, 1
    while 1:
        yield a
        a, b = b, a + b
```

```
Testing:
>>> a = fib()
>>> a.next()
0
>>> for i in range(10):
...     print a.next(),
...
1 1 2 3 5 8 13 21 34 55
```

Here is an iterative algorithm with a memory storage in a list :

```
u=[]
u.append(0)
u.append(1)


for i in range(2,20):
    u.append(u[i-2]+u[i-1])
    print (u[i])
```

Propose a recursive algorithm with a memory storage in a dictionary memo. In term of complexity.

```
<<fibonacci_memo.py>>=
memo = {0:0, 1:1}
```

```
<<fibonacci_memo.py>>=
def fib(n):
.   if not n in memo:
        memo[n] = fib(n-1) + fib(n-2)
  return memo[n]
```

The cover image comes from this site :
https://www.slideshare.net/mrs826/541-interactive-ppt-fibonacci-sequence?next_slideshow=1
that reads about gloden number.
Use python to simulate sequences and test the proporties of Fibonacci sequence and golden number stored in u for instance (like Binet formula https://fr.wikipedia.org/wiki/Suite_de_Fibonacci or https://en.wikipedia.org/wiki/Fibonacci_number )

```
<<fibonacci_binet.py>>=
phi = (1 + 5**0.5) / 2

def fib(n):
    return int(round((phi**n - (1-phi)**n) / 5**0.5))
```

To find the order in the sequence of a Fibonacci number. Find it again using the dictionary memo (like for 144). By the way 144 is the only squared number in the Fibonacci number : can we check it out ?

```
<<fibonacci_binet.py>>=
from math import log

def fibinv(f):
    if f < 2:
        return f
    return int(round(log(f * 5**0.5) / log(phi)))
```

And what about the properties regarding the differences between two consecutive Fibonacci numbers and the gloden ratio.

To help : getting key corresponding to a value can be experimented with
*theKey = [key for (key, value) in memo.items() if value == 144]*

*or*
*from stackoverflow*

```
mydict = {'george':16,'amber':19}
print mydict.keys()[mydict.values().index(16)] # Prints george
```

Or in Python 3.x:

```
mydict = {'george':16,'amber':19}
print(list(mydict.keys())[list(mydict.values()).index(16)]) # Prints george
```