The_millionth_fibonacci_kata

1 The Millionth Fibonacci Kata

In [2]:
```
"""Kata can be found at https://www.codewars.com/kata/the-millionth-fibonacci-kata/tra
grade: 3kyu"""


# Cheekily, this managed to work without timing out: a standard fibonnaci function,
#  and account for negative integers, although it is a very slow method and shouldn't 

def f_sequence(n):
        counter = 1
        a,b = 1,1
        while counter < n-1:
            a, b = a+b, a
            counter += 1
        return a


def fib(n):
    if n == 0:
        return 0
    if n > 0:
        return f_sequence(n)
    else:
        return  int((-1)**(n+1))*f_sequence(abs(n))
```

In [ ]:
```
# annoying Kata due to rounding and precision of higher Fibonnacci terms
```

Using the hint, the re-arranged recurrence relation, one can derive the formula for the negative index:
$F_{n-2} = F_n - F_{n-1}$,
=> $F_{-n} = (-1)^{n+1} F_n$

In [4]:
```
# Using the closed form expression with the golden ratio, won't work due to computatio
```

In [5]:
```
# One can use the 2-dimensional system of linear difference equations describing the F
#therefore starting with  fib(1), fib(0) = 1, 0, yields
```

$$\begin{pmatrix} F_{k+2} \\ F_{k+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{k+1} \\ F_k \end{pmatrix}$$
$$=> \begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

```
In [7]: # and, therefore the problem can be solved with matrix exponentiation,
```

$$=> \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n} = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

From Wikipedia : Using the identities below, one can calculate F_n recursively in O(log(n)) arithmetic operations and in time O(M(n) log(n)), where M(n) is the time for the multiplication of two numbers of n digits. This matches the time for computing the nth Fibonacci number from the closed-form matrix formula, but with fewer redundant steps if one avoids recomputing an already computed Fibonac ci number (recursion with memoization), so using M(n) wiht np and recursion with memoization would be fastest with this method:

$$F_{2n-1} = F_n^2 + F_{n-1}^2$$
$$F_{2n} = (F_{n-1} + F_{n+1})F_n$$
$$= (2F_{n-1} + F_n)F_n.$$

```
In [9]: def matmult(a,b):  # Np .dot or matmult cause rounding errors, so bespoke matrix multi
            zip_b = list(zip(*b))  # would be more efficient to calculate A_11, A_22, A_21 sep
            return [[sum(ele_a*ele_b for ele_a, ele_b in zip(row_a, col_b))
                     for col_b in zip_b] for row_a in a]
        def fib(n):
            if n == 0:
                return 0
            if n < 0:
                return  int((-1)**(n+1))*fib(abs(n))  # using formula for negative index
            A = [[1,1],[1,0]]  # (([F_n+1,F_n],[F_n,F_n-1]))
            X = [[1],[0]] # Calculated upon to form ([F_n+1, F_n])
            # iterate through calculating A*n.X
            while n > 0:
                if n % 2 == 0:
                    # using power rule, M^(2n) = M^(2)^(n)
                    A = matmult(A,A)
                    n /= 2
                else:
                    X = matmult(A,X)
                    n -= 1
            return X[1][0] # Fn
```

```
In [10]: # Essentially, in pseudocode the above fib function calculates:
```

$$\vec{X}_n = \mathbf{A}^n \vec{X}_0$$

```
In [12]: # by
```

while n > 0:
if n=2m is even:

$\mathbf{A}^{2m} = (\mathbf{A}^2)^m$
n => m , A => A^2
and if n=2m+1 is odd:
$\mathbf{A}^{2m+1}\vec{X}_l = \mathbf{A}^{2m}\vec{X}_{l+1}$
n => 2m , $\vec{X}_l => \vec{X}_{l+1}$