Sam Traylor

Carola Wenk

CMPS 1500

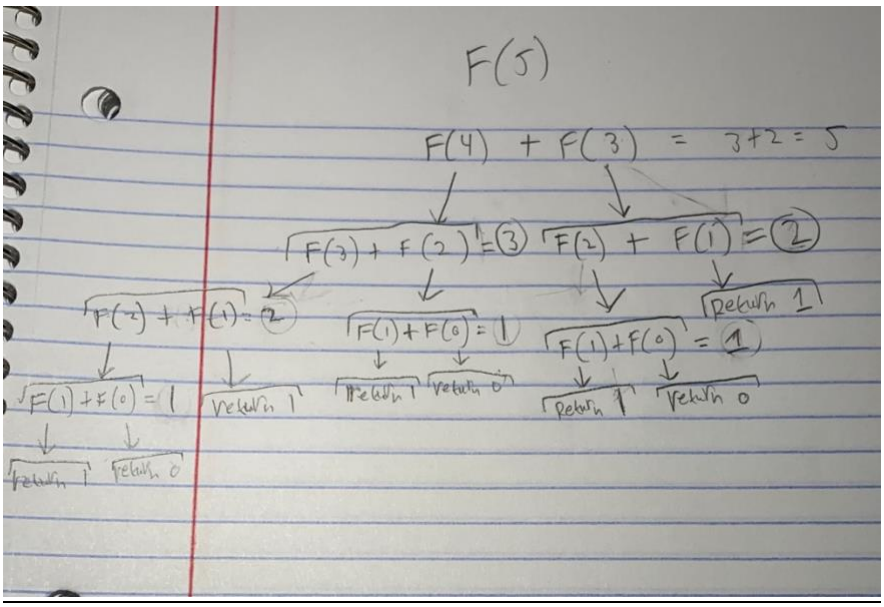
3 April 2019

Lab 7 Problem 2

Code to return nth Fibonacci number both recursively (F(n)) and Iteratively (f(n))

```
def F(n):
    '''
    Code to recursively find and return the n-th number in the fibbonaci sequence

    Keyword arguments:
    n -- an integer representing how deep you want the fibonacci sequence to go

    '''
    if n == 0: #two-part base case
        return 0
    if n == 1:
        return 1
    else:
        return F(n-1) + F(n-2) #recursive case


def f(n):
    '''
    Code to find and return n-th fibonacci number using iteration

    Keyword arguments:
    n -- an integer representing how deep you want the fibonacci sequence to go

    '''
    fibonacci = [0,1]
    x = 2
    while len(fibonacci) != n + 1:
        fibonacci.append(fibonacci[x-1] + fibonacci[x-2])
        x += 1
    return fibonacci[-1]
```

Trace of call to recursive function F(5):



Runtime Comparison of Recursive and Iterative Fibonacci Programs

Input Number (N)	Recursive program time	Iterative program time
10	0.030819	0.034486
20	0.052916	0.040651
30	0.44073	0.036182
40	42.600006	0.041368

For smaller input sizes, the efficiency of recursion shows, because it's faster at the beginning than using traditional loop structure, but that changes with the input increasing. As the input approaches 40 the time It takes to compute recursively shoots up exponentially, whereas the loop structure time is still very reasonable. This means that there's a time and place for each structure: sometimes recursion will be faster, but for some problems and some input sizes, iteration is faster.