Fibonacci sequence with Python recursion and memoization

Article · June 2019

CITATIONS
0

READS
1,300

1 author:

Kelvin Wangonya Kinyanjui
Kabarak University
1 PUBLICATION **0** CITATIONS

SEE PROFILE

Fibonacci sequence with Python recursion and memoization

June 16, 2019

The Fibonacci sequence is a sequence of numbers such that any number, except for the first and second, is the sum of the previous two. For example:

$$0, 1, 1, 2, 3, 5, 8, 13, 21...$$

> **Fun fact**: November 23 is celebrated as Fibonacci day because when the date is written in the mm/dd format (11/23), the digits in the date form a Fibonacci sequence: 1,1,2,3.

We can represent this in the formula:

$$fib(n) = fib(n-1) + fib(n-2)$$

With this formula, we can write a simple recursive function to solve for `fib(n)` .

Note: Only use this to test for small numbers, preferably n < 10. I'll explain later

```
def fib(n):
    if n < 2:  # base case
        return n
```

```
    return fib(n-1) + fib(n-2)

if __name__ == "__main__":
  print(fib(7))
```

Why do we need a base case?

It's easy enough to convert the formula directly into

```
def fib(n):
    return fib(n-1) + fib(n-2)
```

The problem with this though is that when you run it, it throws a `RecursionError`. This simply means that there's possibly an infinite loop in the program. A base case in a recursive function tells the function when to stop (to avoid going into an infinite loop) and is usually something that is already known or that can be solved for easily without needing the function. In our case here, we know from the definition that any number in the sequence, **except for the first and second**, is the sum of the previous two. We use that to form our base case `if n < 2: return n`.

Making it more efficient

Remember when I told you to only test the program with small values of `n`? Here's why.

As it stands, every call to `fib()` results in two more calls to `fib()` in the `return` statement. The call tree grows exponentially. 15 calls are required to compute `fib(5)`, 177 calls for `fib(10)`, 21,891 for `fib(20)` ... you get the point. To solve this problem, we can use memoization.

Memoization helps avoid unnecessary calculation of the same values if they were already previously calculated. It works just like memorization

for humans. You already have `2 x 2` memorized and can give the answer immediately without having to use a calculator. `799 x 377` ? You probably need to use a calculator for that. If, for some reason, you find that you get asked `799 x 377` a lot, it would be nice to have it memorized so you don't have to calculate it every other time. The value of `799 x 377` will always remain the same, so all you have to do is calculate it once, save the value in your "cache" (memory), and retrieve it every time you need it.

Luckily, python has a built-in decorator that does just that.

```
 from functools import lru_cache

@lru_cache(maxsize=None)
def fib(n):
    if n < 2:  # base case
        return n
    return fib(n-1) + fib(n-2)

if __name__ == "__main__":
  print(fib(50))
```

Now you can test with large numbers safely.

<<·Home·>>