Memoization

Memoization ensures that a function doesn't run for the same inputs more than once by keeping a record of the results for the given inputs (usually in a dictionary).

For example, a simple recursive function for computing the nth Fibonacci number:

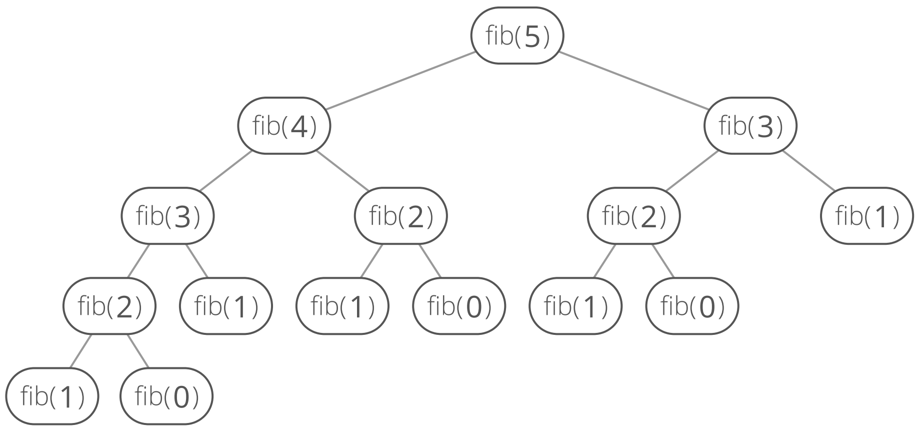
Python 3.6 ▾

```
def fib(n):
    if n < 0:
        raise IndexError(
            'Index was negative. '
            'No such thing as a negative index in a series.'
        )
    elif n in [0, 1]:
        # Base cases
        return n

    print("computing fib(%i)" % n)
    return fib(n - 1) + fib(n - 2)
```

Will run on the same inputs multiple times:

```
>>> fib(5)
computing fib(5)
computing fib(4)
computing fib(3)
computing fib(2)
computing fib(2)
computing fib(3)
computing fib(2)
5
```

We can imagine the recursive calls of this function as a tree, where the two children of a node are the two recursive calls it makes. We can see that the tree quickly branches out of control:



To avoid the duplicate work caused by the branching, we can wrap the function in a class with an attribute, `memo`, that maps inputs to outputs. Then we simply

1. check `memo` to see if we can avoid computing the answer for any given input, and
2. save the results of any calculations to `memo`.

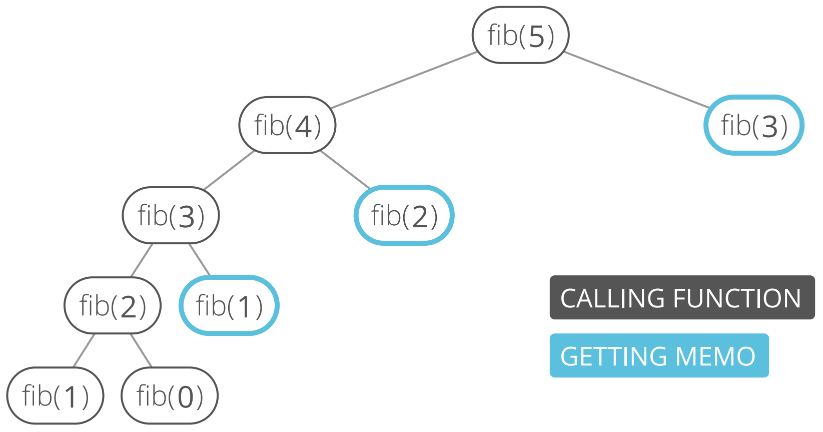
Python 3.6 ▾

```python
class Fibber(object):

    def __init__(self):
        self.memo = {}

    def fib(self, n):
        if n < 0:
            raise IndexError(
                'Index was negative. '
                'No such thing as a negative index in a series.'
            )

        # Base cases
        if n in [0, 1]:
            return n

        # See if we've already calculated this
        if n in self.memo:
            print("grabbing memo[%i]" % n)
            return self.memo[n]

        print("computing fib(%i)" % n)
        result = self.fib(n - 1) + self.fib(n - 2)

        # Memoize
        self.memo[n] = result

        return result
```

We save a bunch of calls by checking the memo:

```
>>> Fibber().fib(5)
computing fib(5)
computing fib(4)
computing fib(3)
computing fib(2)
grabbing memo[2]
grabbing memo[3]
5
```

Now in our recurrence tree, no node appears more than twice:



Memoization is a common strategy for **dynamic programming** problems, which are problems where the solution is composed of solutions to the same problem with smaller inputs (as with the Fibonacci problem, above). The other common strategy for dynamic programming problems is **going bottom-up (/concept/bottom-up)**, which is usually cleaner and often more efficient.

← course home (/table-of-contents)

Next up: Bottom-Up Algorithms ➡ (/concept/bottom-up?course=fc1&section=dynamic-programming-recursion)