The Fibonacci Numbers Algorithm with Memoization

In this lesson, we will employ memoization in the Fibonacci numbers algorithm.

We'll cover the following

Optimizing Fibonacci number's algorithm

Let's revisit the Fibonacci numbers algorithm from an earlier lesson of this course.

```
def fib(n):
  if n == 0: # base case 1
    return 0
  if n == 1: # base case 2
    return 1
  else: # recursive step
    return fib(n-1) + fib(n-2)

print (fib(10))
```

We have also reproduced a dry run of `Fib(6)` below to visualize how this algorithm runs.

Evaluate Fib(6)

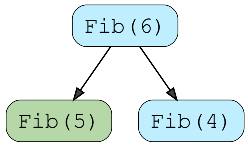
1 of 46

Evaluate Fib(6)

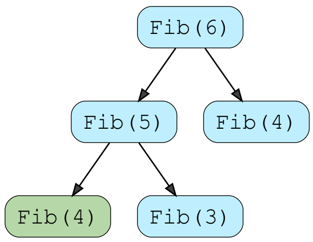
Fib(6)

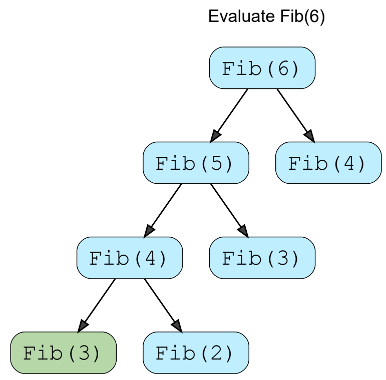
2 of 46

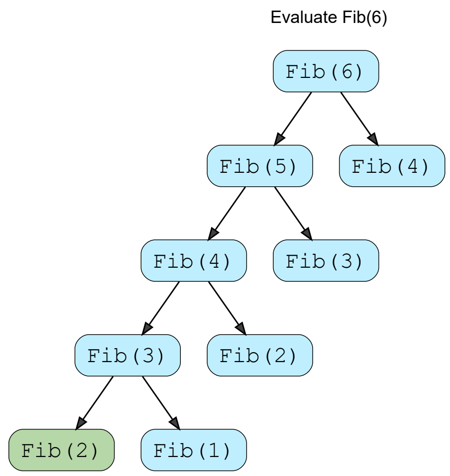
Evaluate Fib(6)



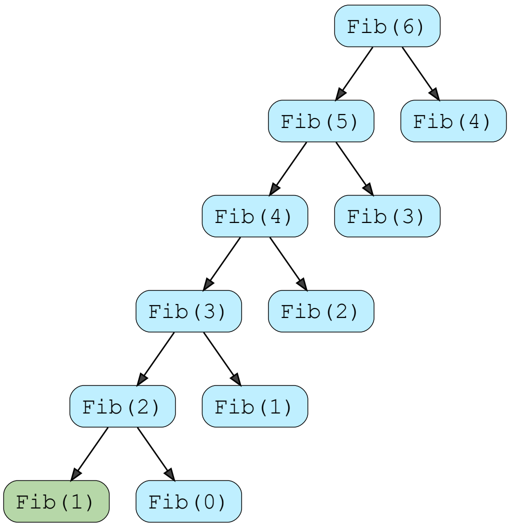
Evaluate Fib(6)



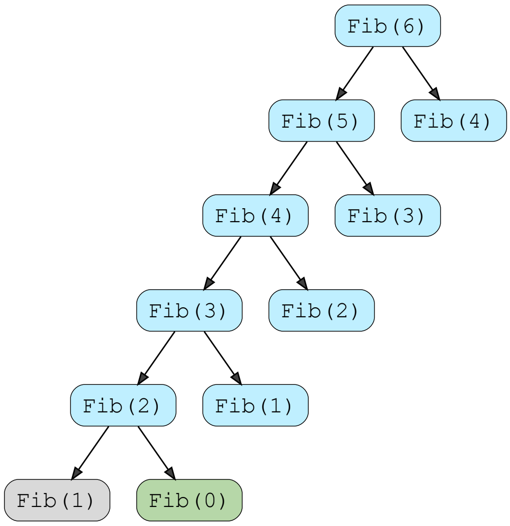

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)


Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)

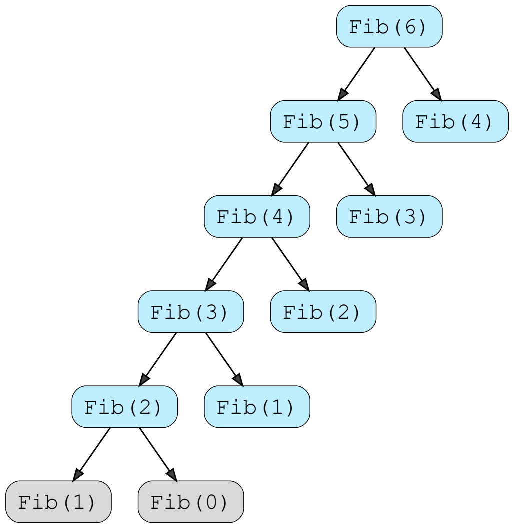
Evaluate Fib(6)



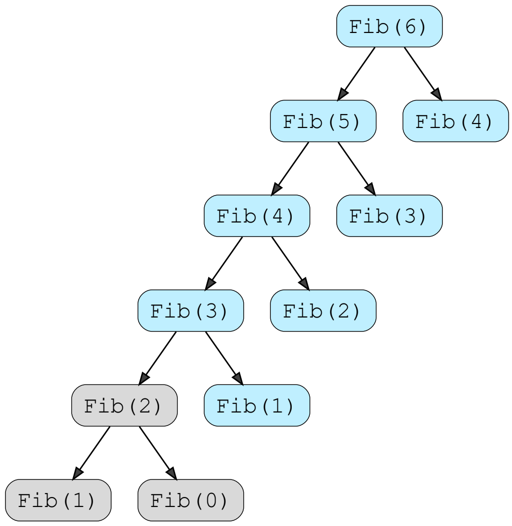
Evaluate Fib(6)

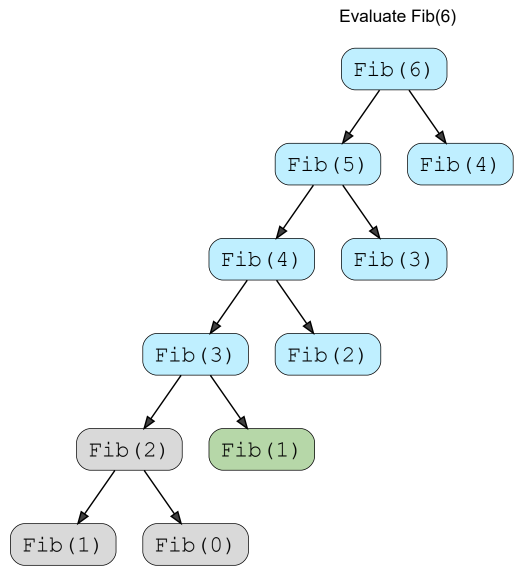


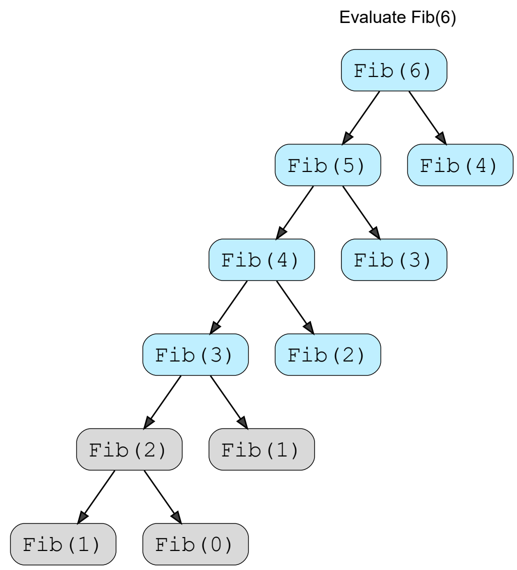
Evaluate Fib(6)

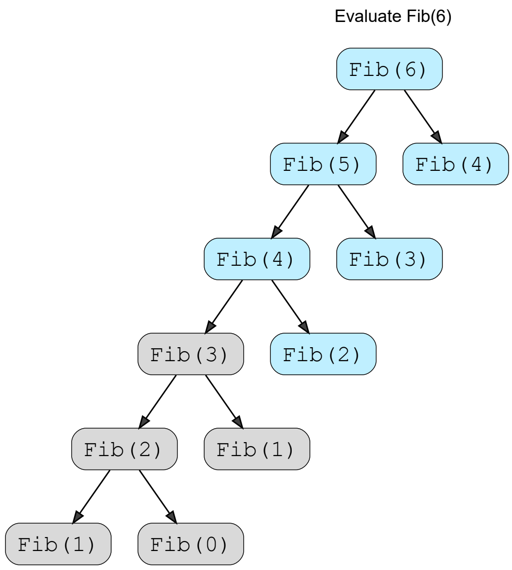


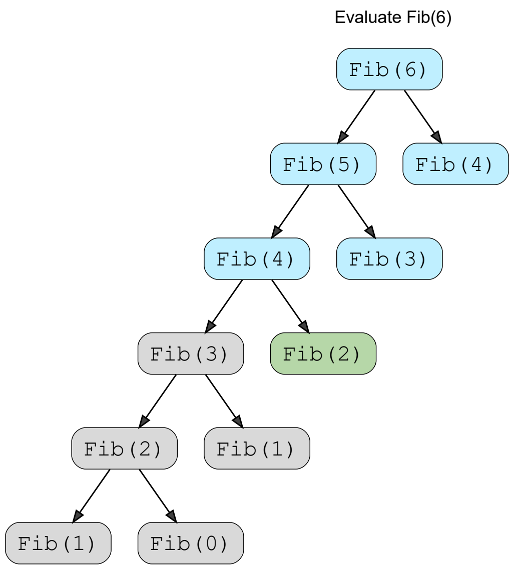
Evaluate Fib(6)



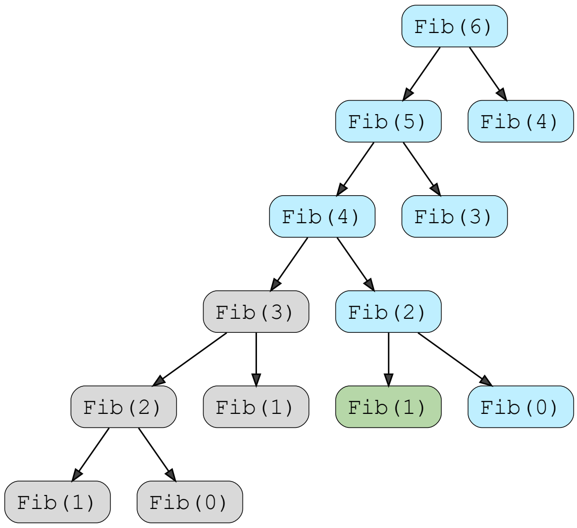

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)


Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)

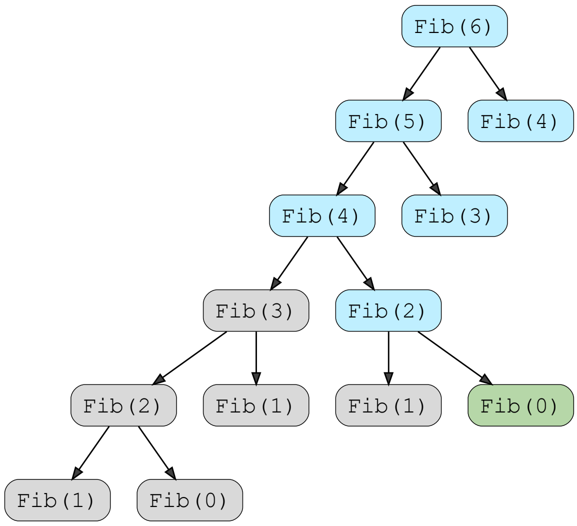

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)

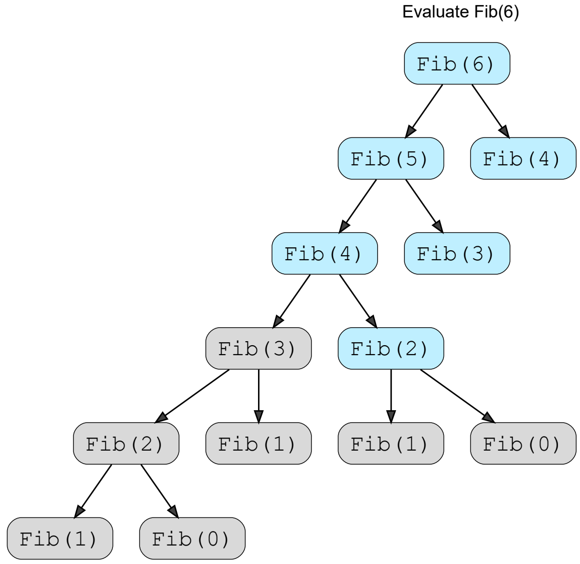

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)

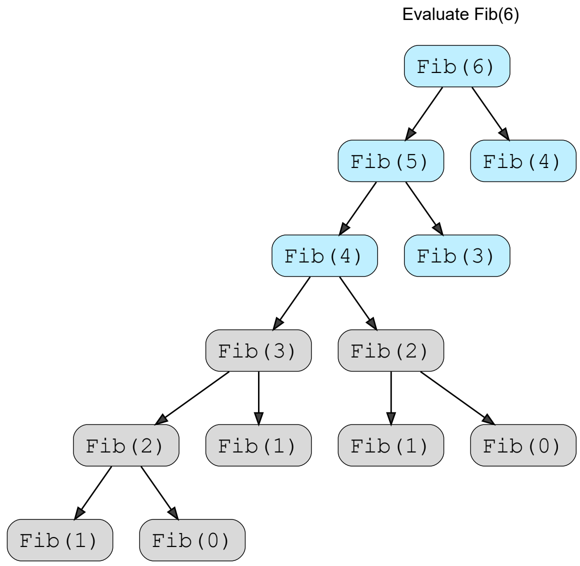
Evaluate Fib(6)

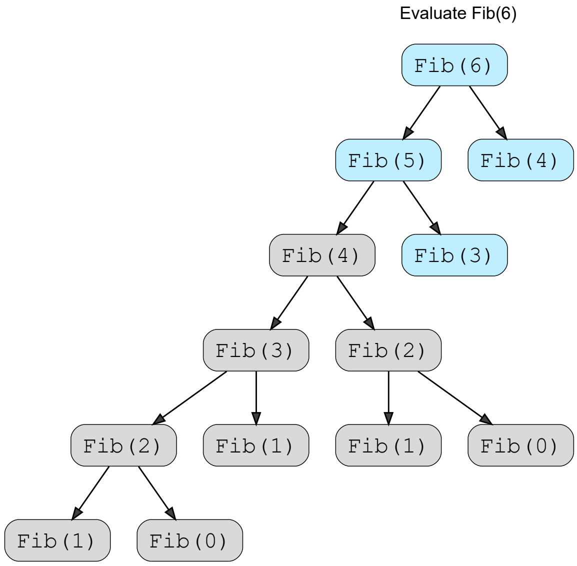


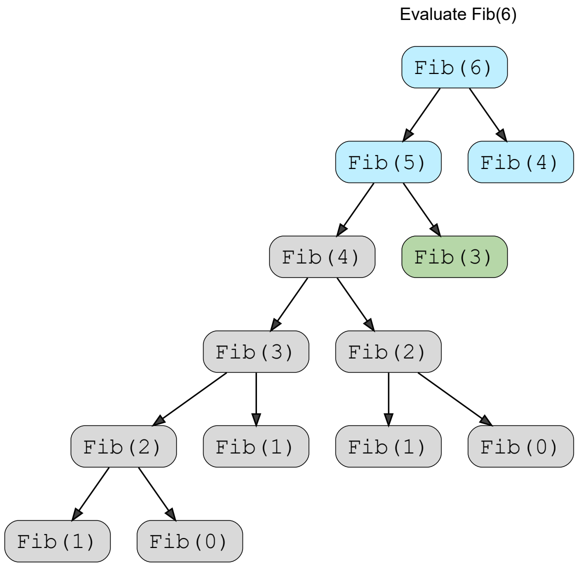
Evaluate Fib(6)

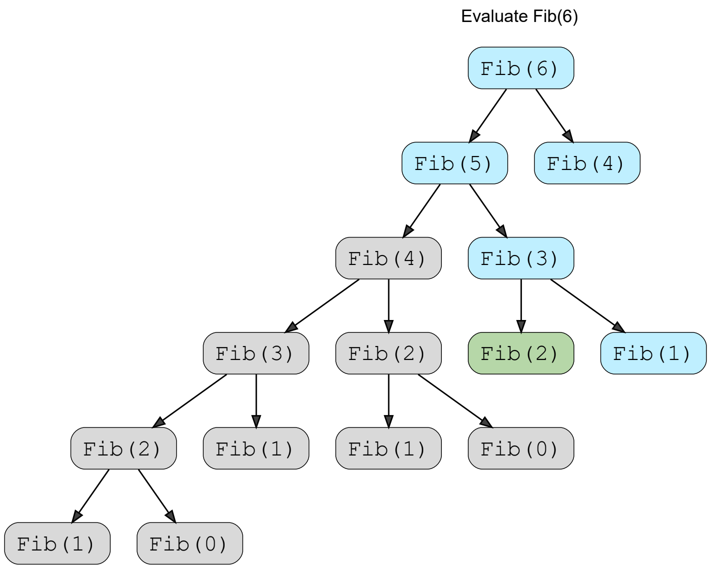


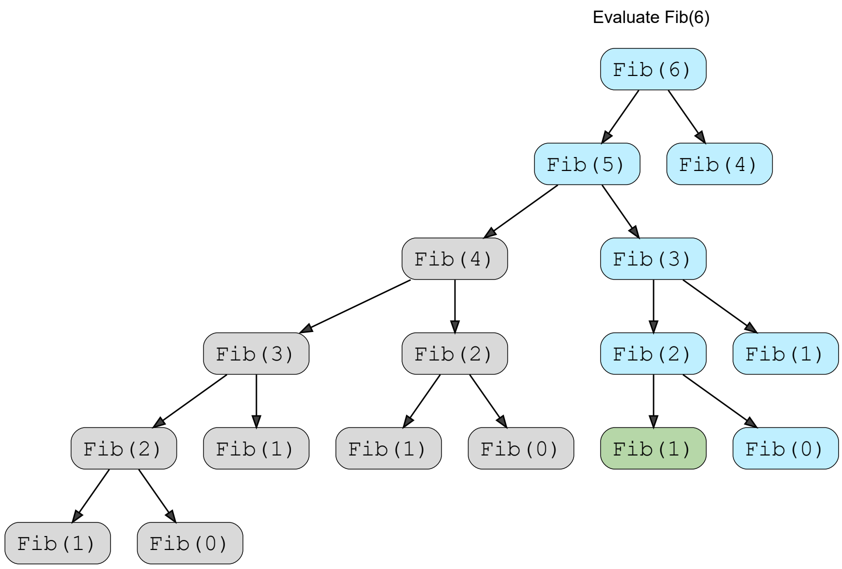

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

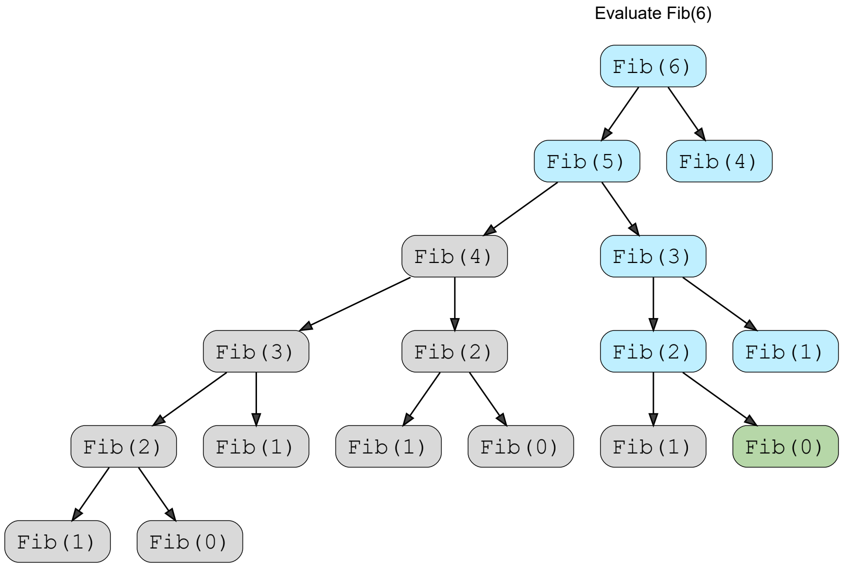

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

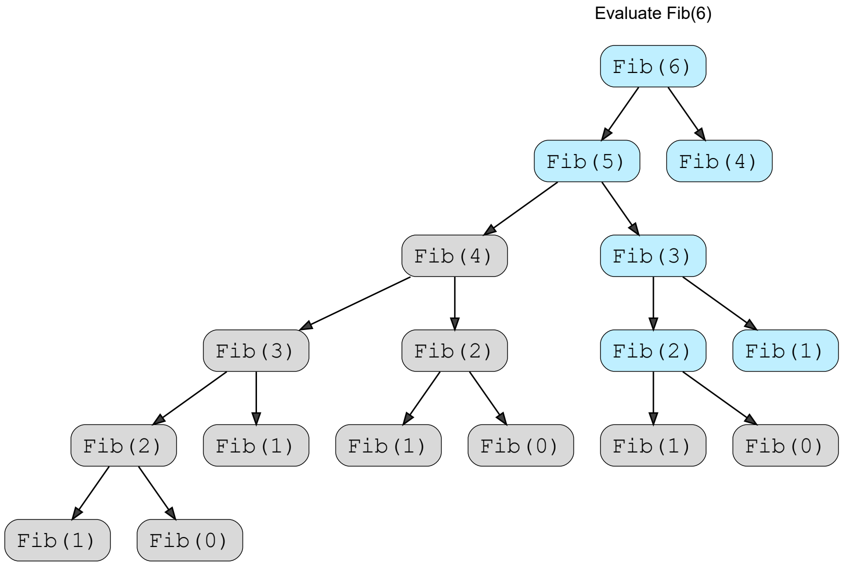

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)


Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

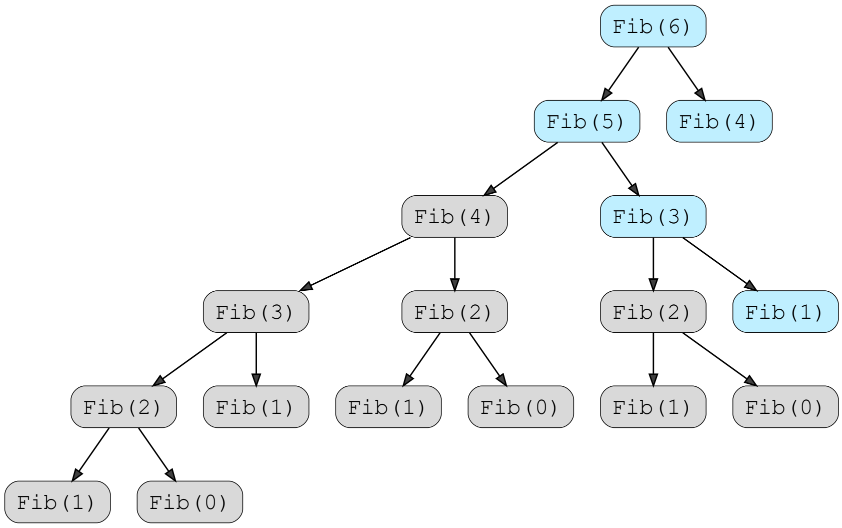

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)


Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

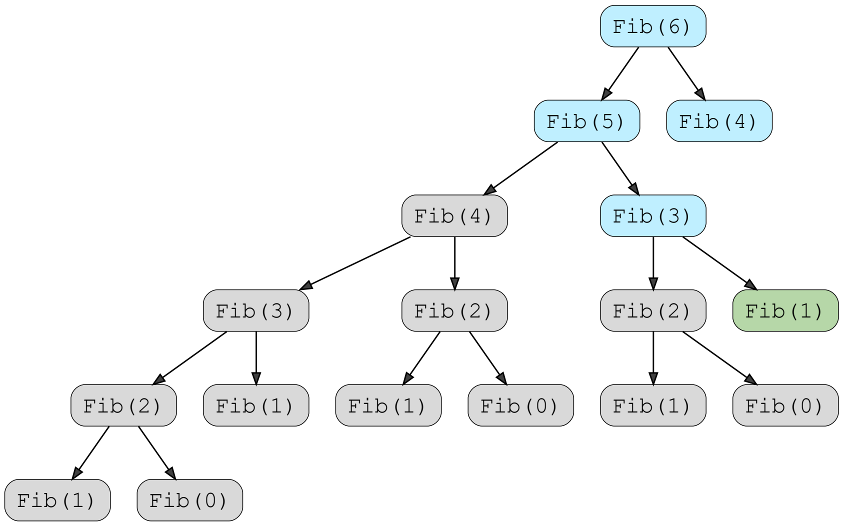

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

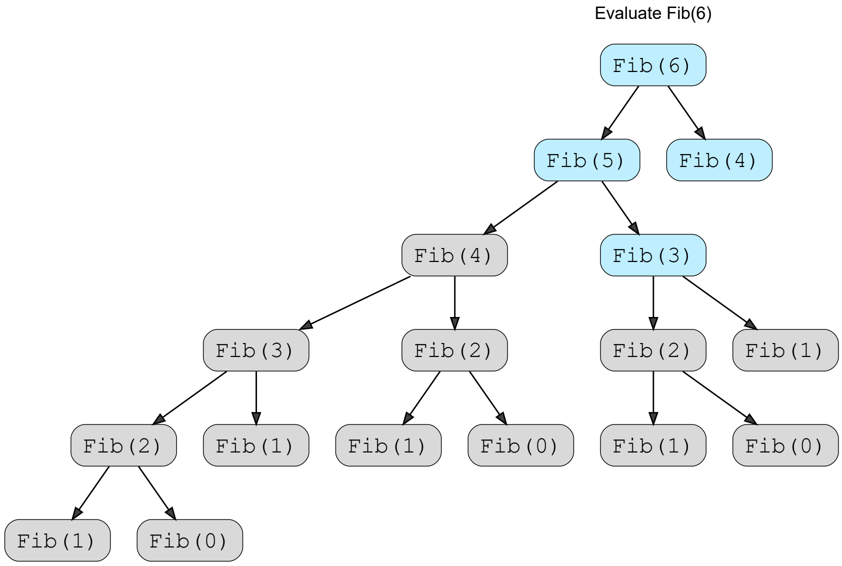

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

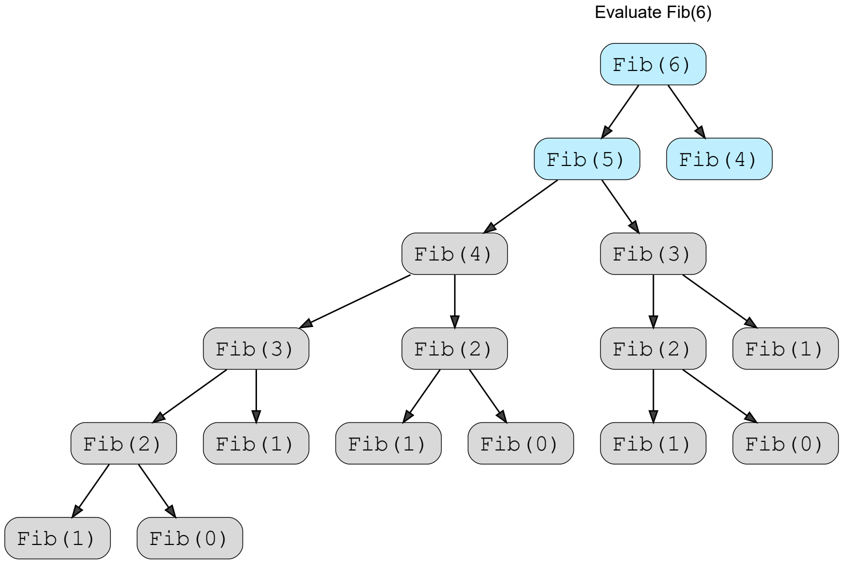
Evaluate Fib(6)



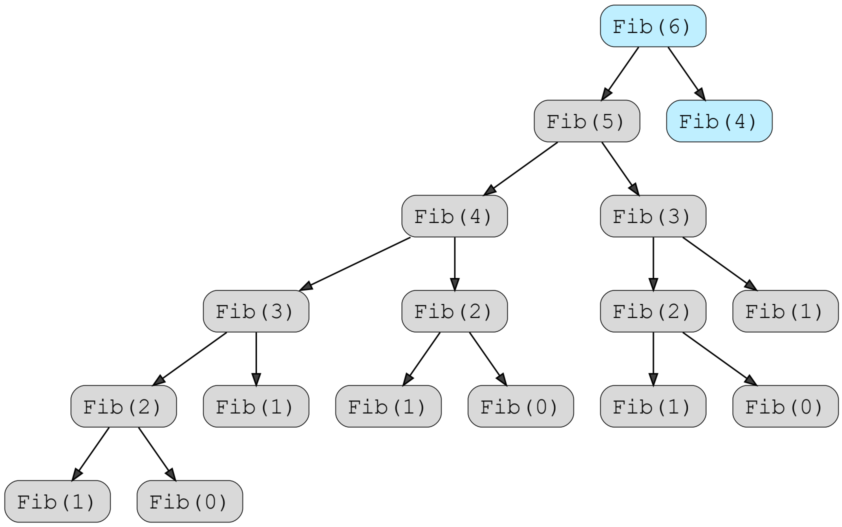
Evaluate Fib(6)



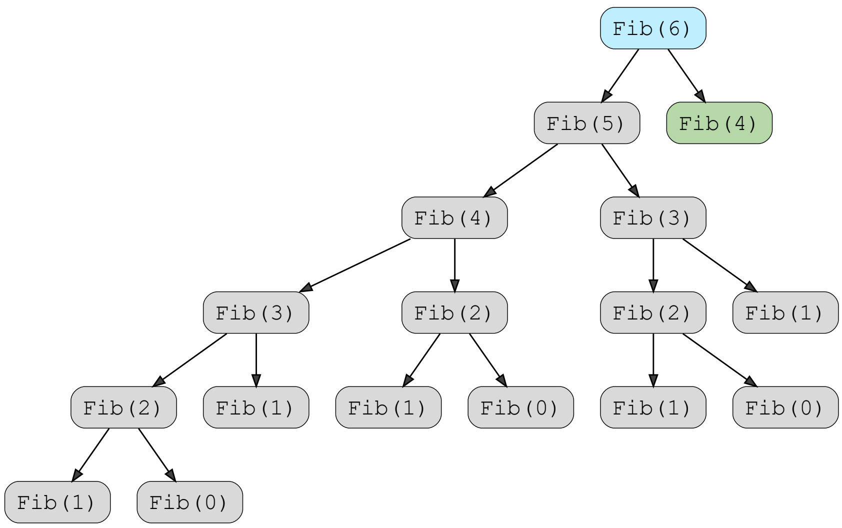

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)


Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

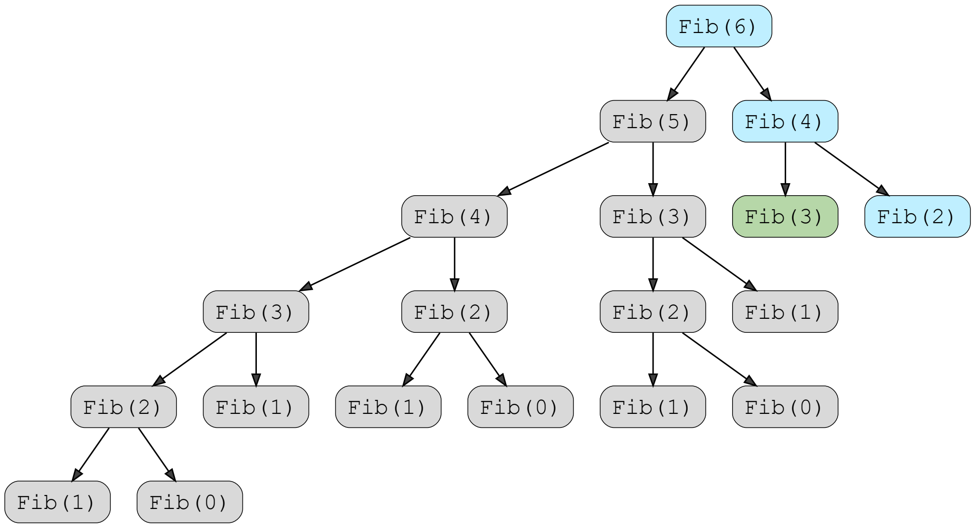
Evaluate Fib(6)



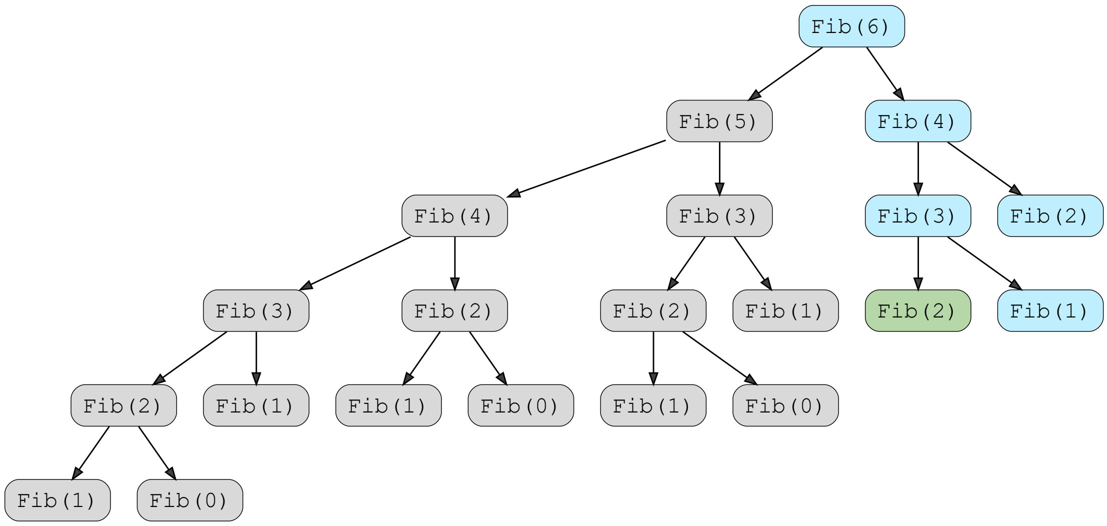
Evaluate Fib(6)

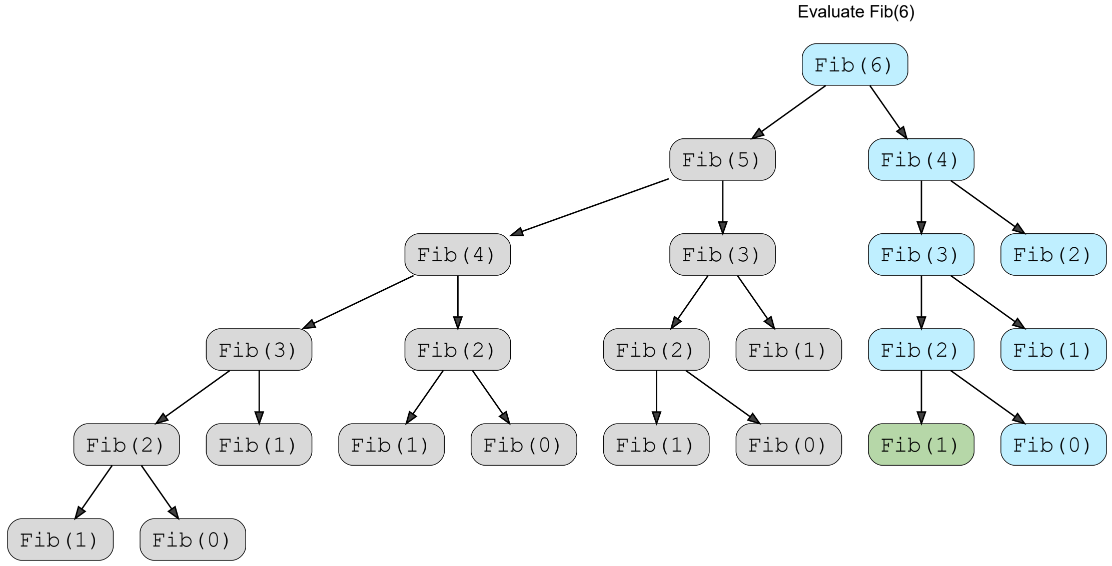


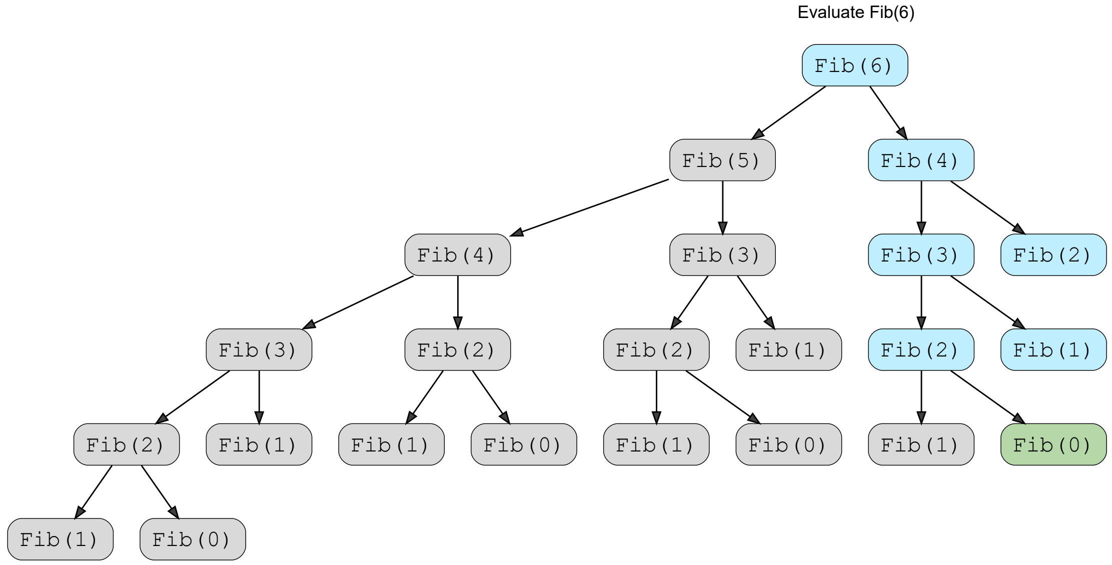
Evaluate Fib(6)

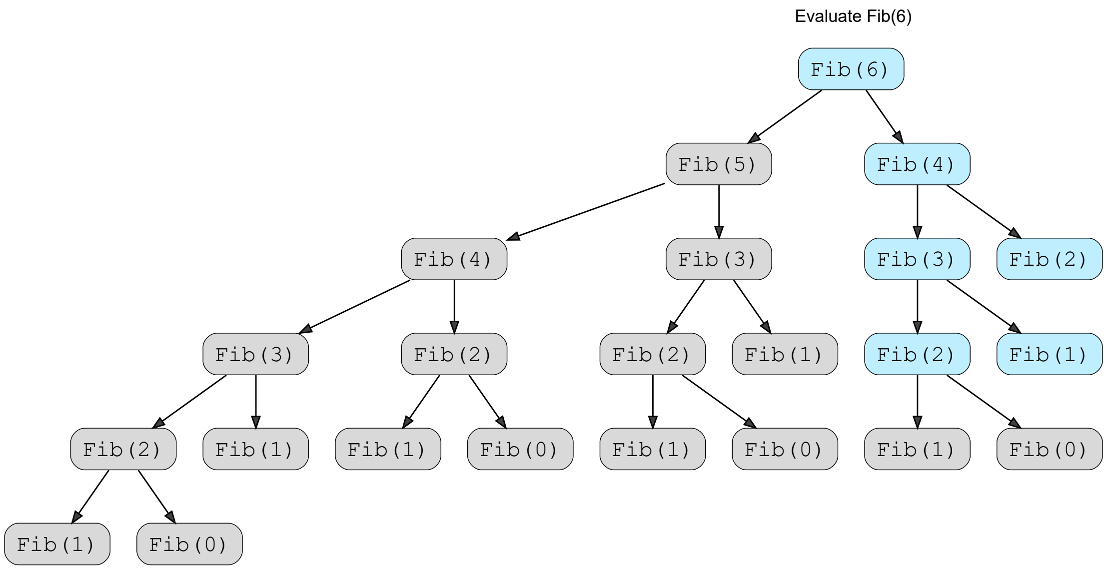


Evaluate Fib(6)

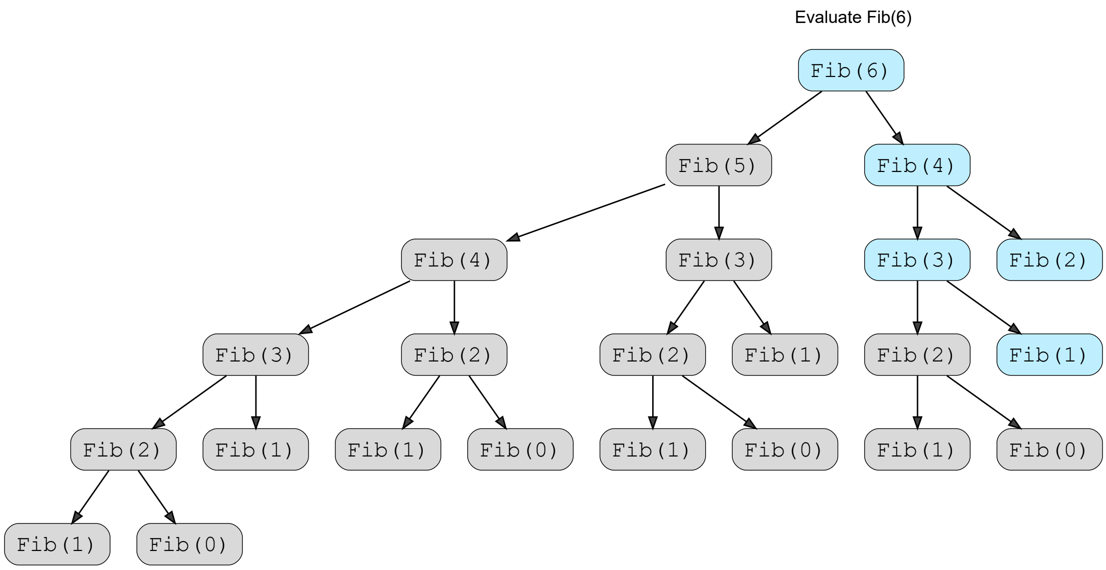



Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)


Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

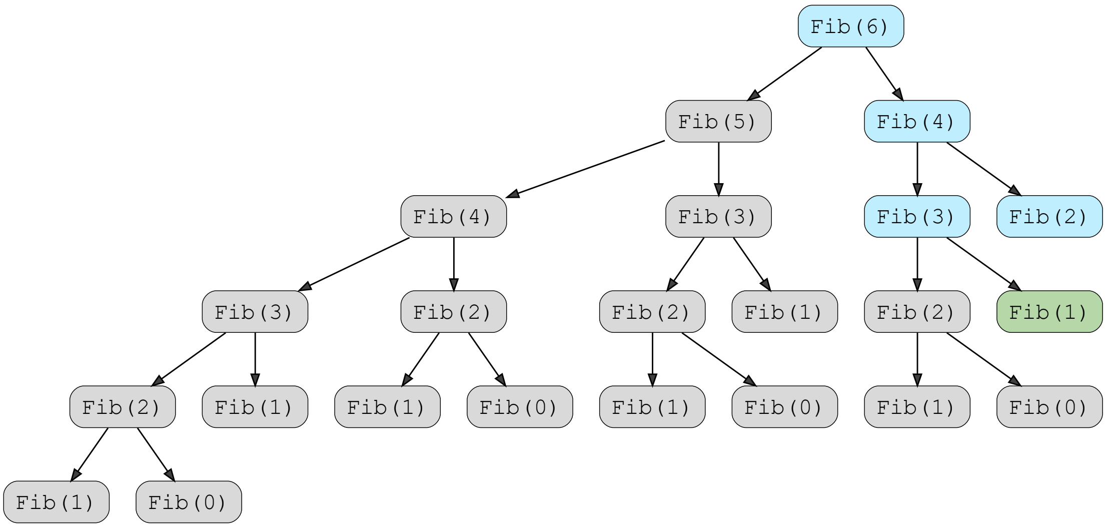


35 of 46

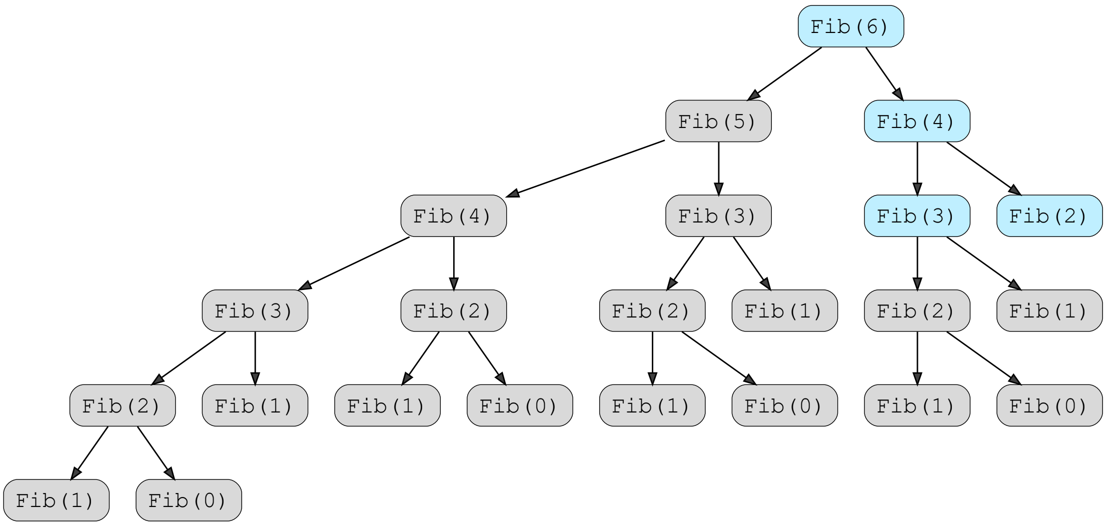


36 of 46

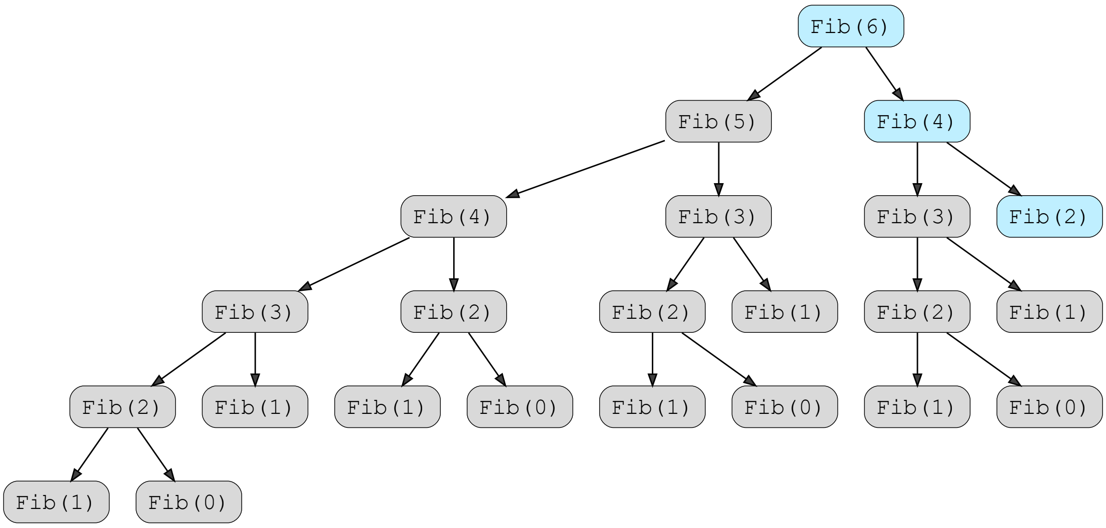
Evaluate Fib(6)



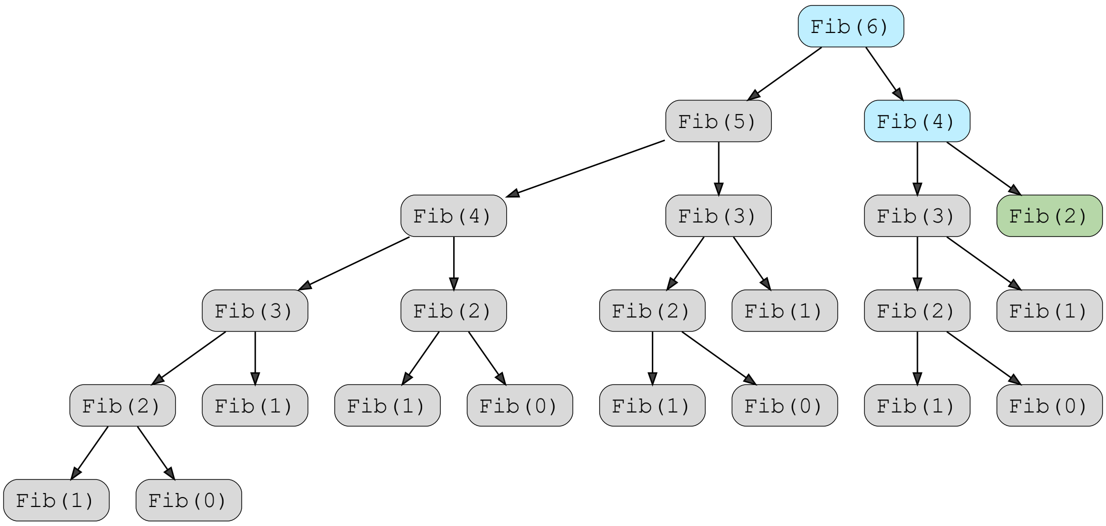
Evaluate Fib(6)

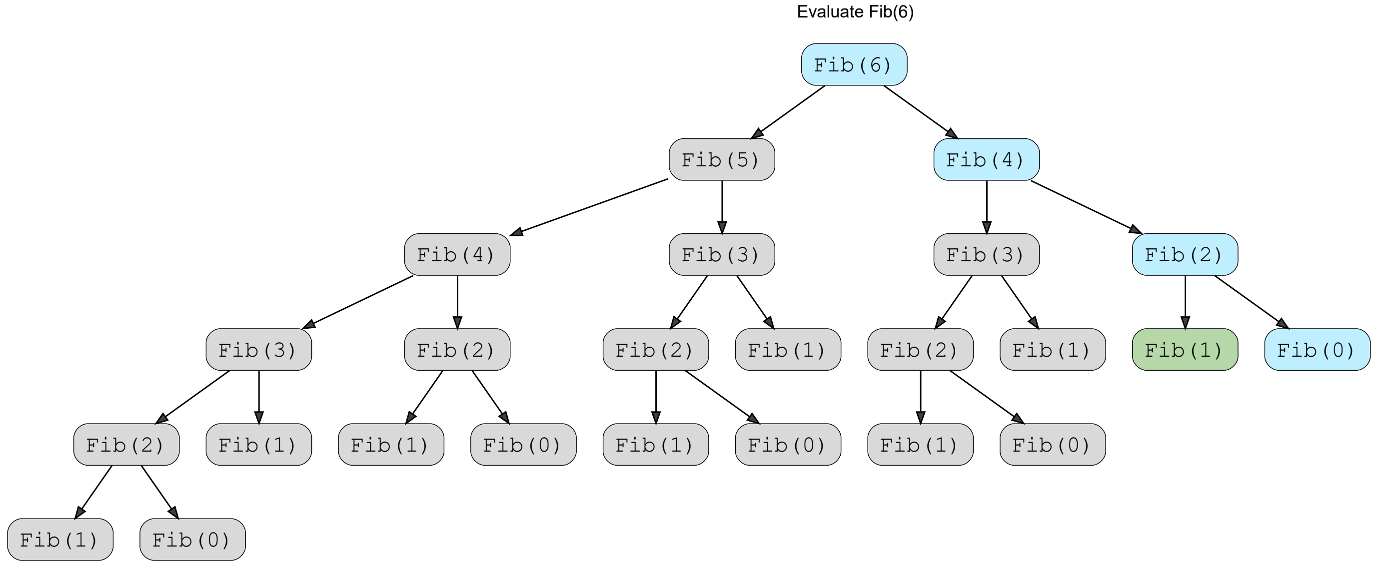


Evaluate Fib(6)

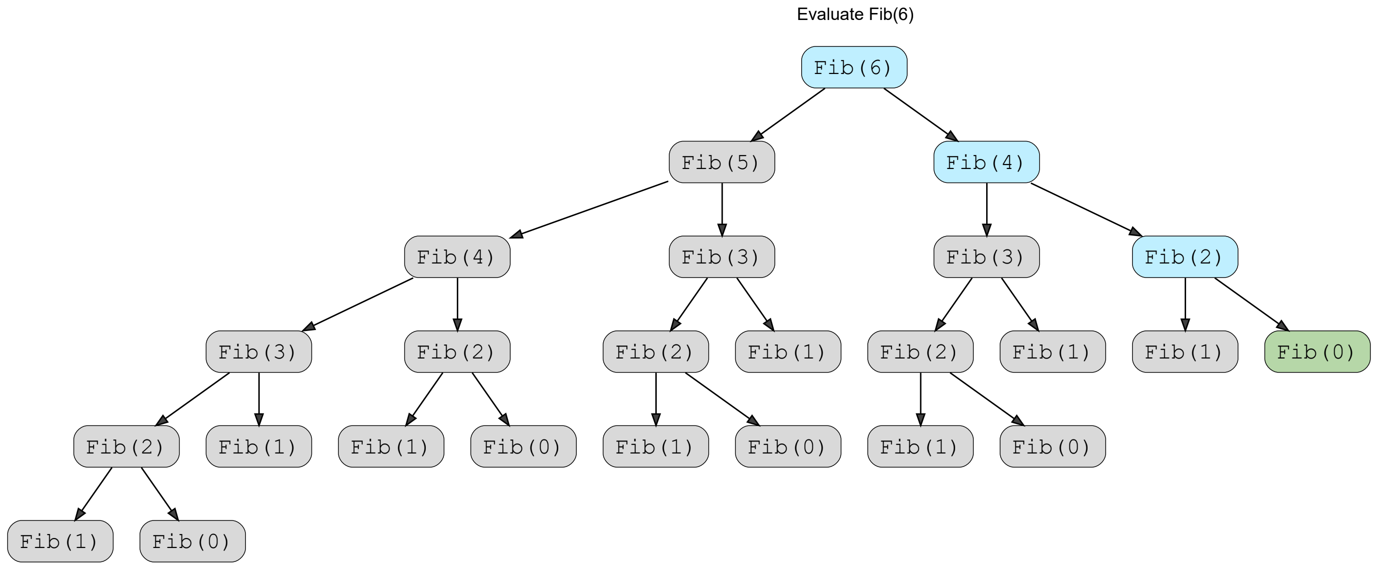


Evaluate Fib(6)



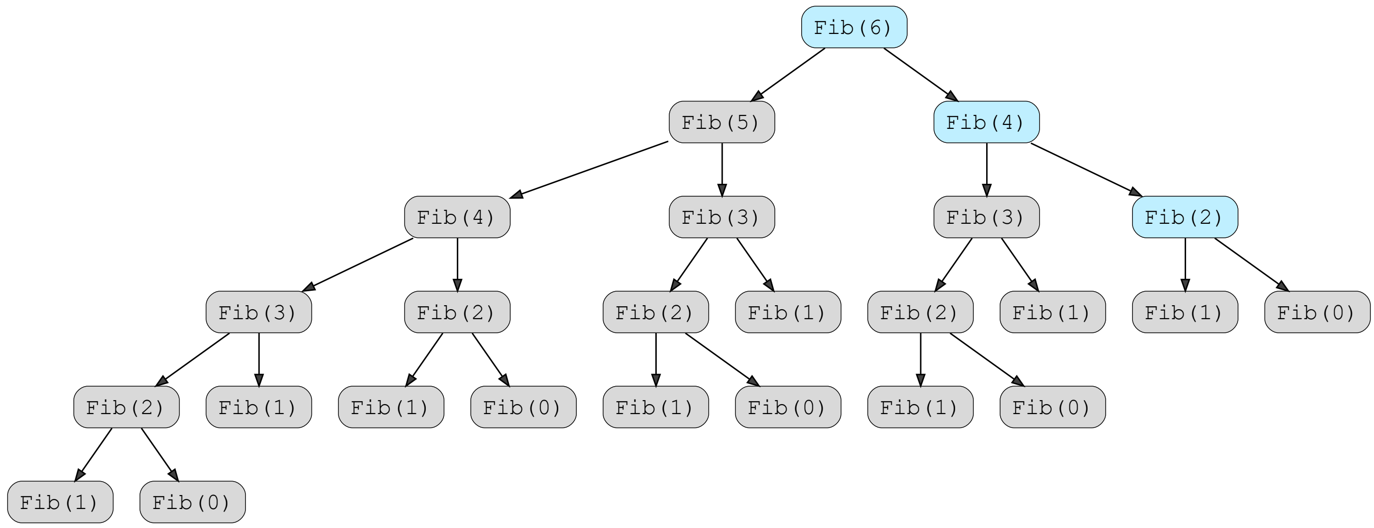


41 of 46

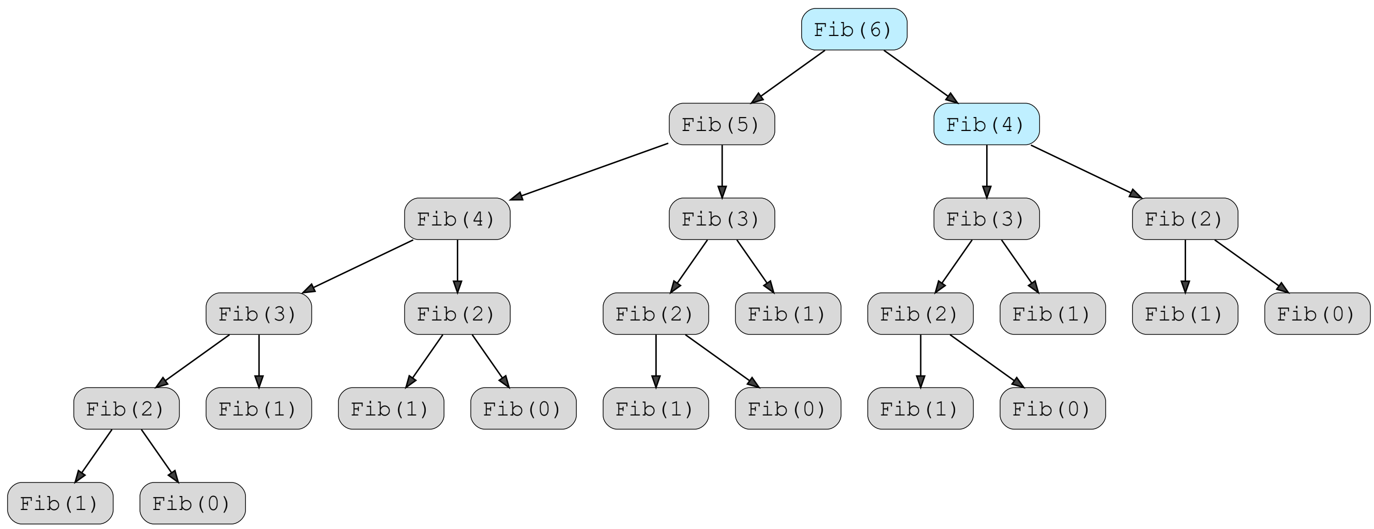


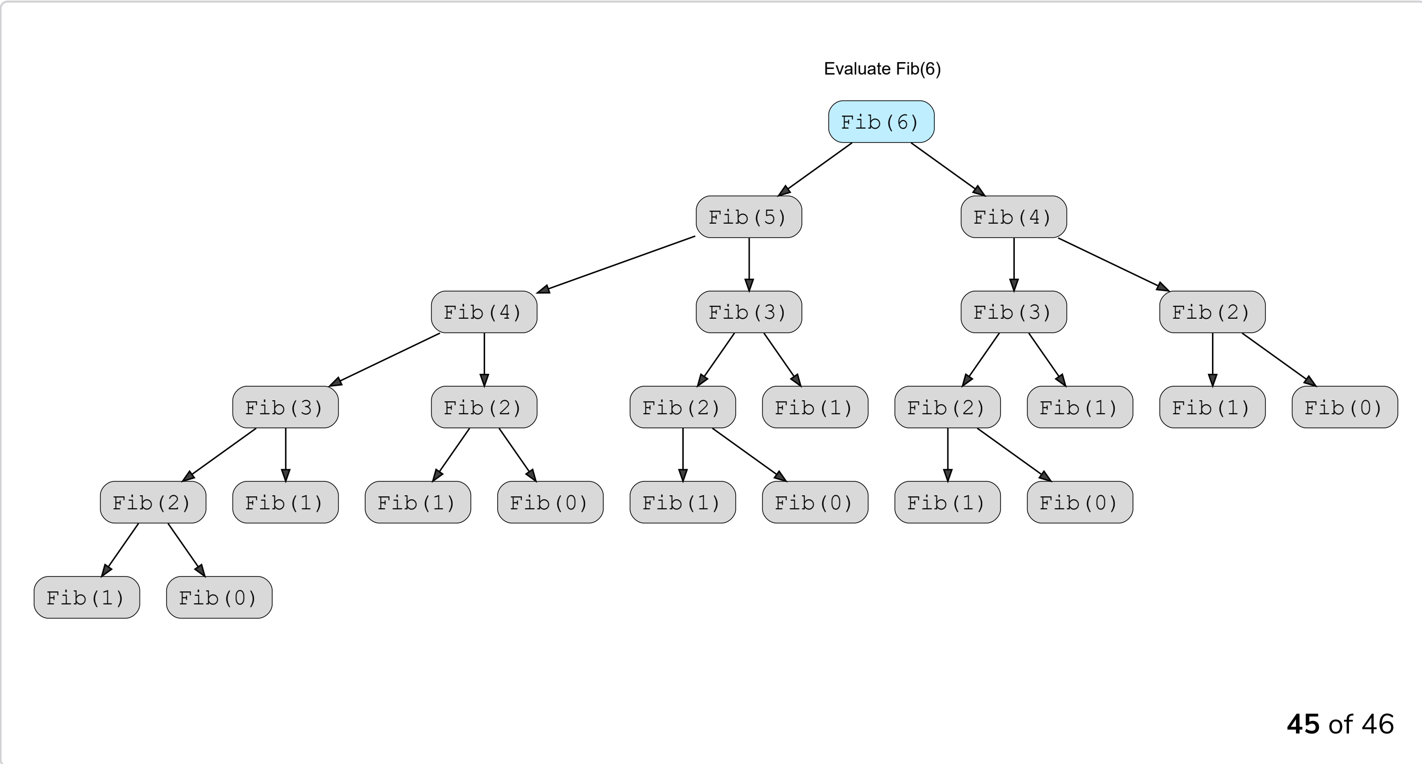
42 of 46

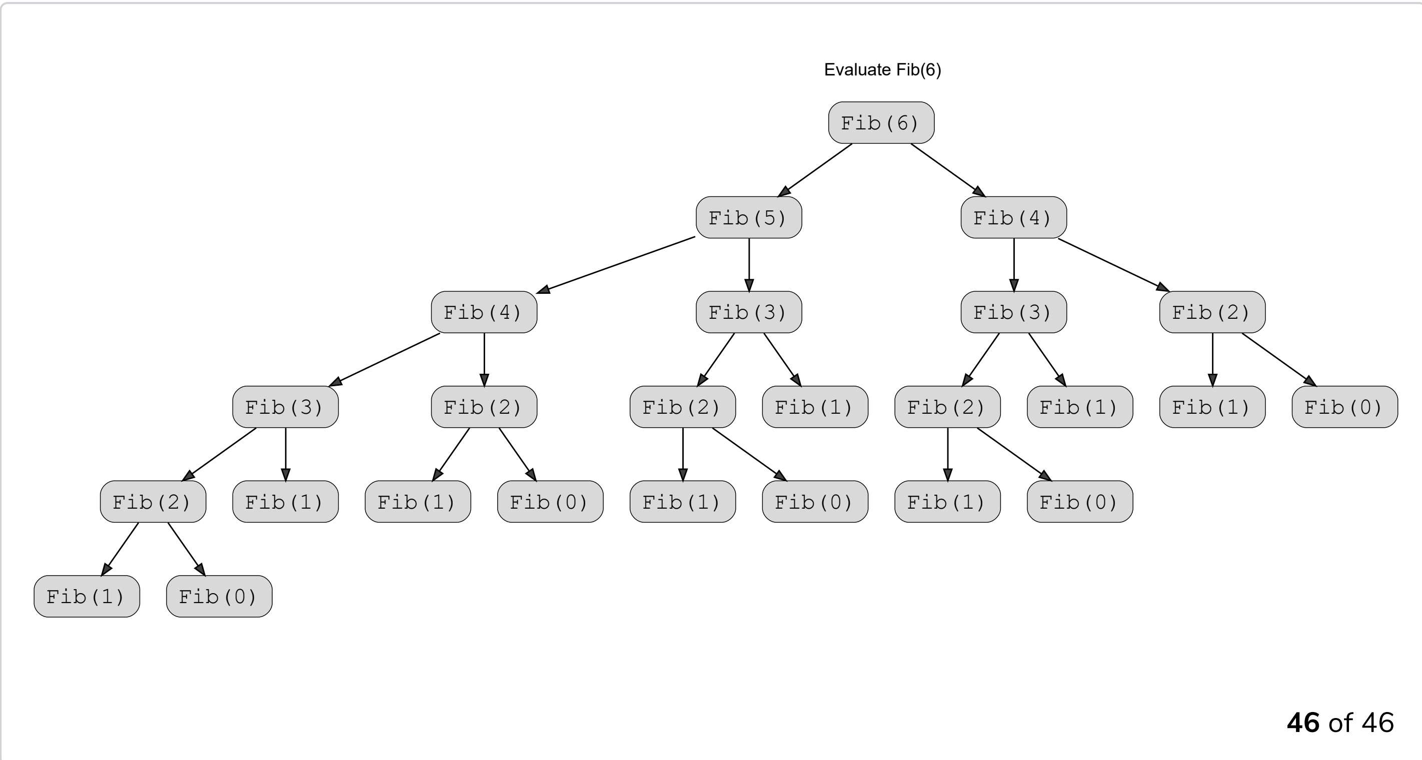
Evaluate Fib(6)



Evaluate Fib(6)







Memoization

Now let's store the results of every evaluated Fibonacci number and reuse it whenever it is needed again. We can use either a list or dictionary for memoization in this problem. In this course, we will use a dictionary because it is more convenient than a list. The only changes we are going to make to our original Fibonacci algorithm is the addition and usage of this new dictionary. We have defined a dictionary globally so it is available to all the `fib()` calls (*line 1*). Next, after the base cases, we check whether this number has already been evaluated, if it was, its memoized value is returned (*lines 8-9*). Otherwise, we need to evaluate this result, and this is where we make the recursive call (*lines 11-12*). Here before returning, we memoize the result in `memo[n]`. We do not need to memoize base cases as they are already usually *O(1)* operations.

```
memo = {} #dictionay for Memoization

def fib(n):
  if n == 0: # base case 1
    return 0
  if n == 1: # base case 2
    return 1
  elif n in memo: # Check if result for n has already been evaluated
    return memo[n] # return the result if it is available
  else: # otherwise recursive step
    memo[n] = fib(n-1) + fib(n-2) # store the result of n in memoization dictionary
    return memo[n] # return the value

print (fib(100))
```

Time complexity

Notice how you can now run much bigger numbers as well. This is the benefit of memoization; now we do not need to recalculate `fib()` calls because their results have been stored through memoization. This reduces the time complexity of our algorithm from **$O(2^n)$** to **O(n)**. This is because to evaluate `fib(n)` we need the result of `fib(n-1)`, and `fib(n-2)`, `fib(n-2)` would already be evaluated from `fib(n-1)`'s recursive call, thus its value will be available to `fib(n)` in *O(1)*. If you continue this, you will see how the second recursive call for all the subsequent calls is always *O(1)*. Only the first call will need to reach the base case, which will make it a simple linear structure instead of a tree. Look at the following visualization.

Evaluate Fib(6)

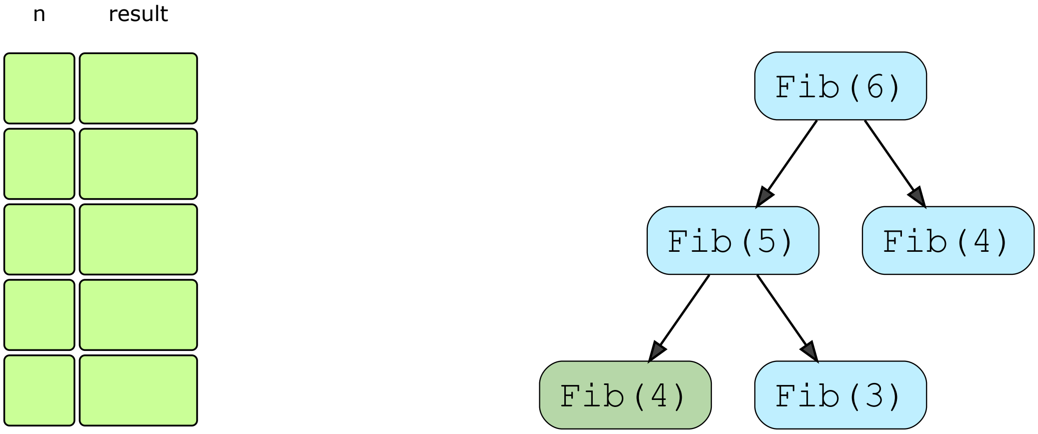
Evaluate Fib(6)

Result not memoized, make recursive call

Evaluate Fib(6)

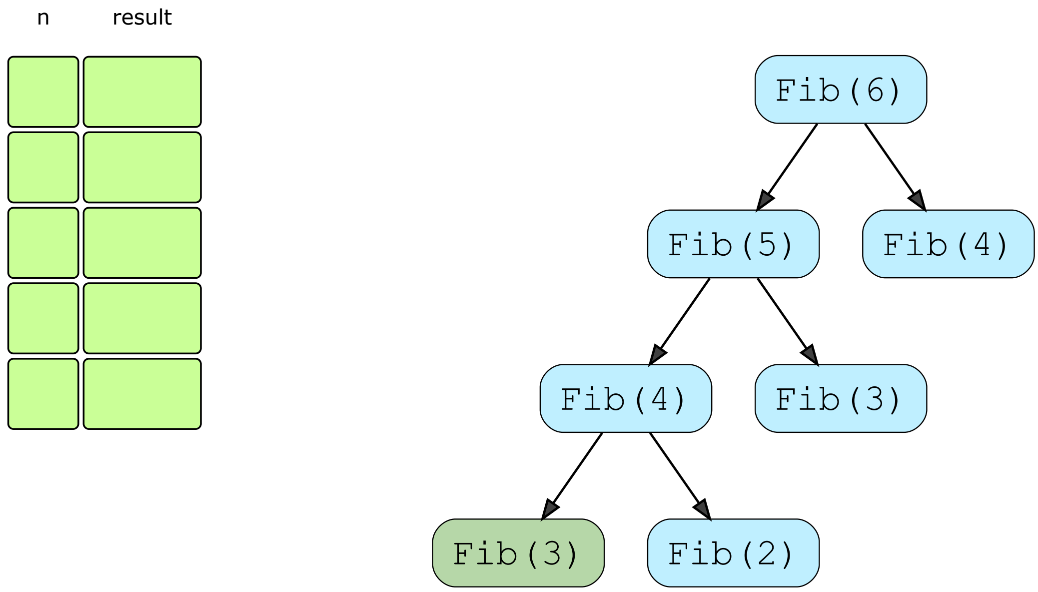
Result not memoized, make recursive call

Evaluate Fib(6)



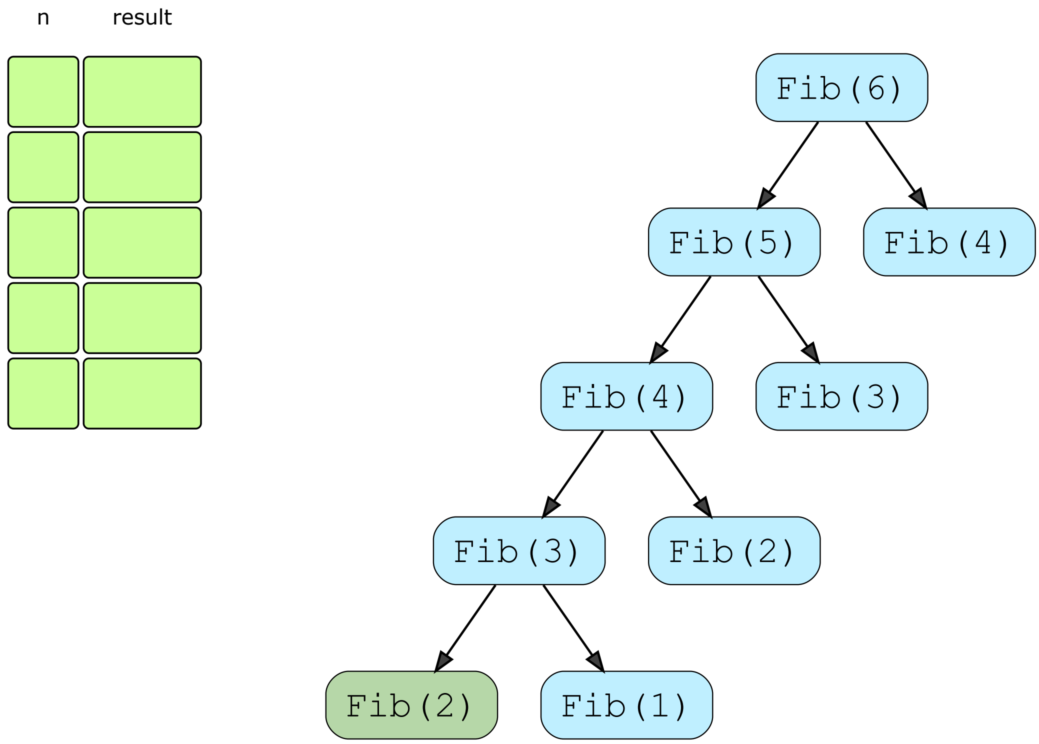
Result not memoized, make recursive call

Evaluate Fib(6)



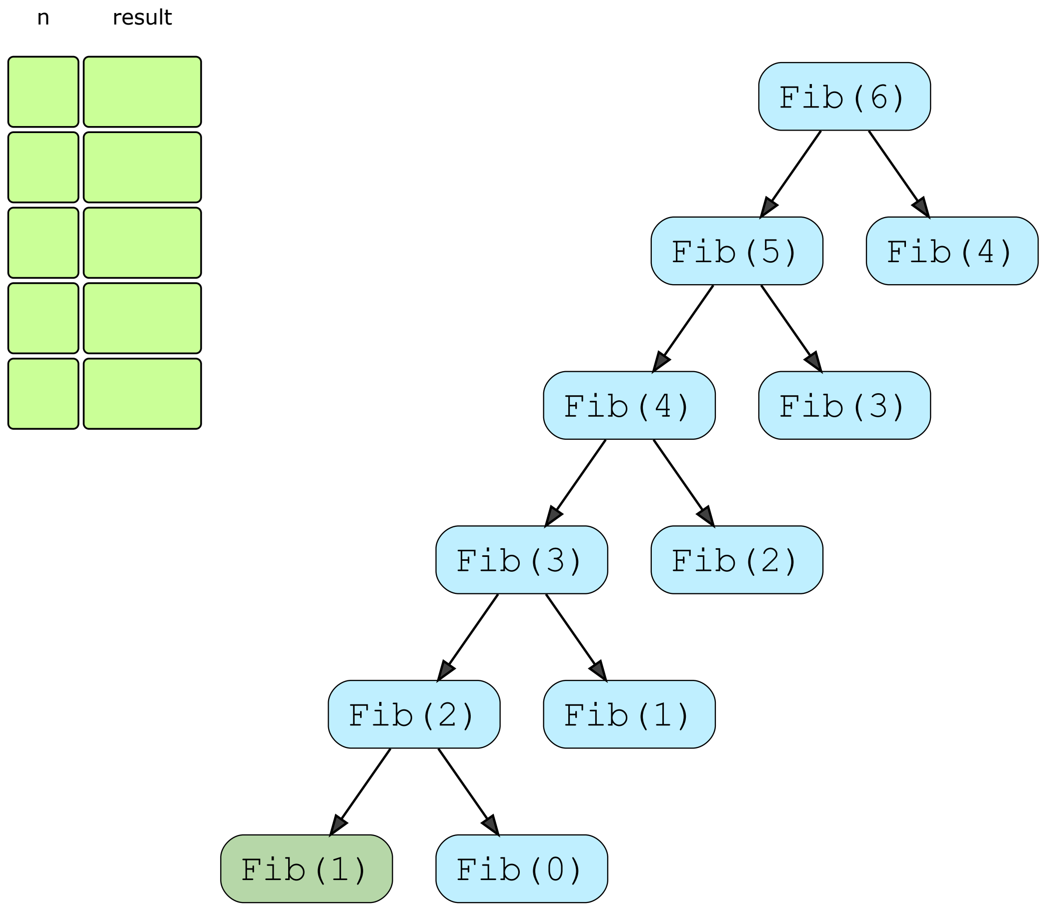
Result not memoized, make recursive call

Evaluate Fib(6)



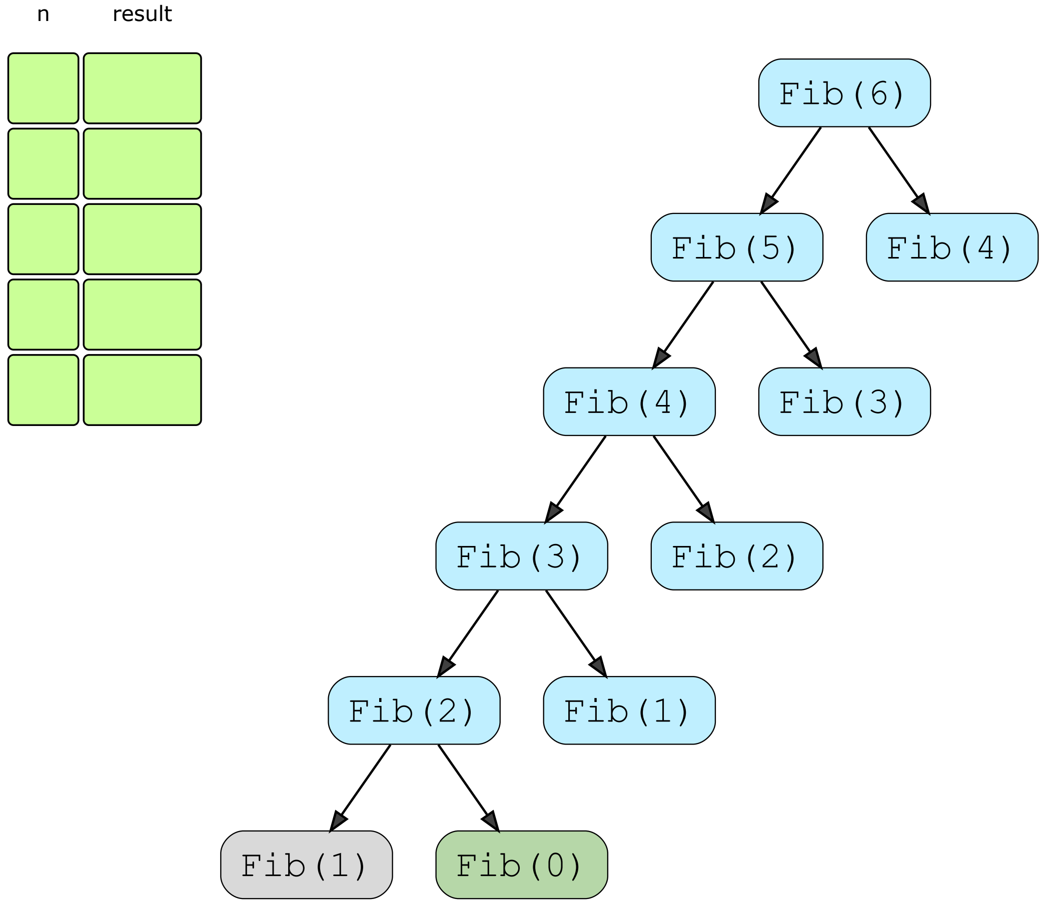
Result not memoized, make recursive call

Evaluate Fib(6)



base case

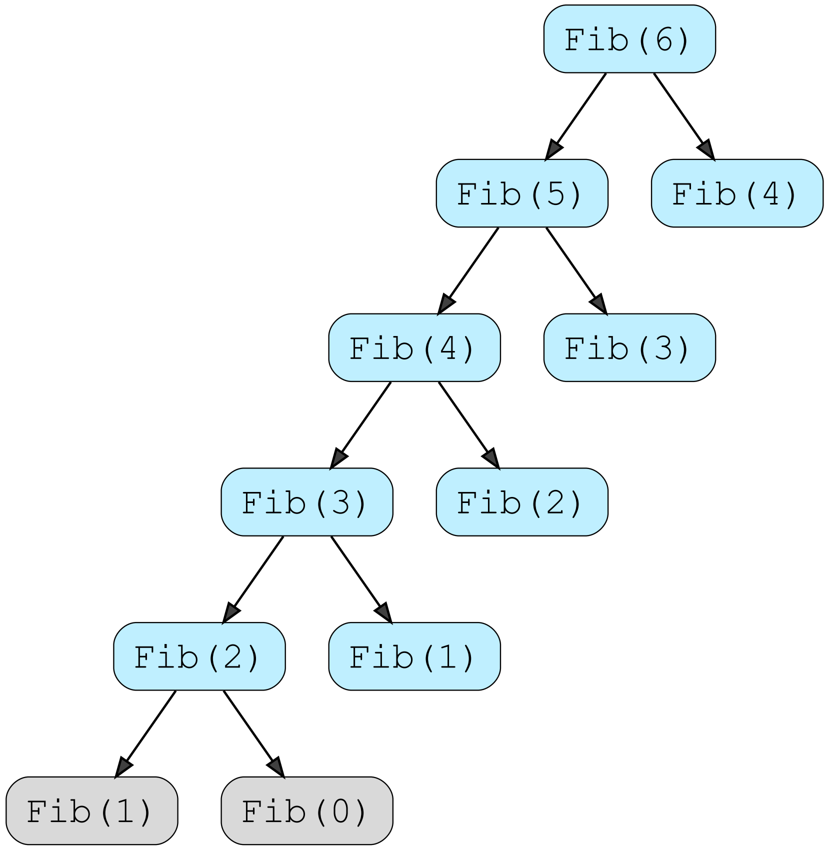
Evaluate Fib(6)



base case

Evaluate Fib(6)

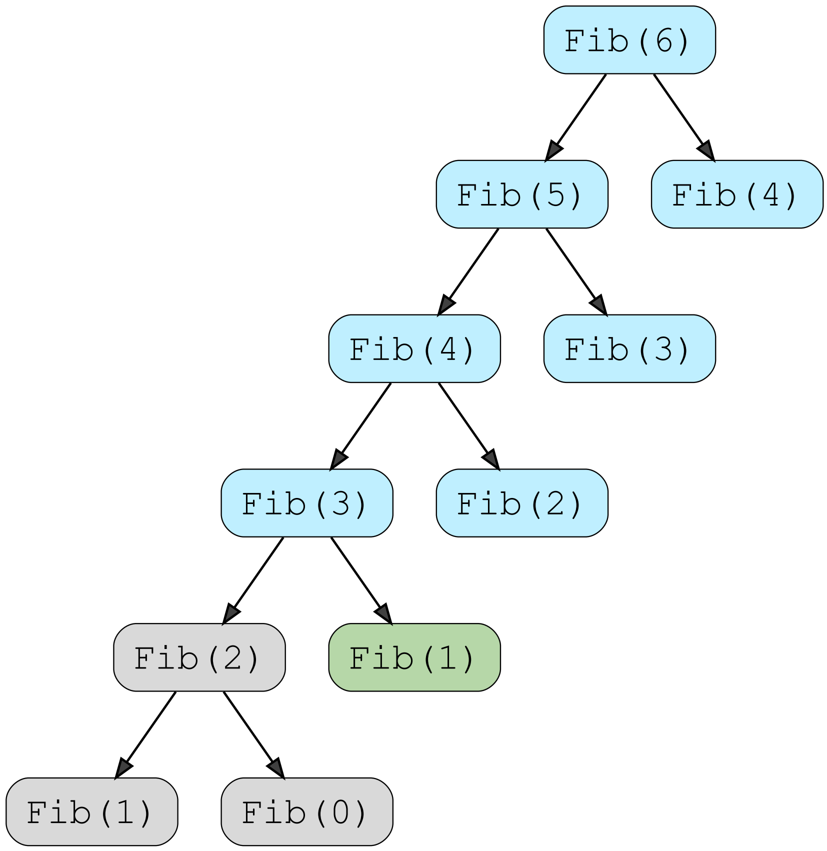
n	result
2	1



Result evaluated, memoize the result

Evaluate Fib(6)

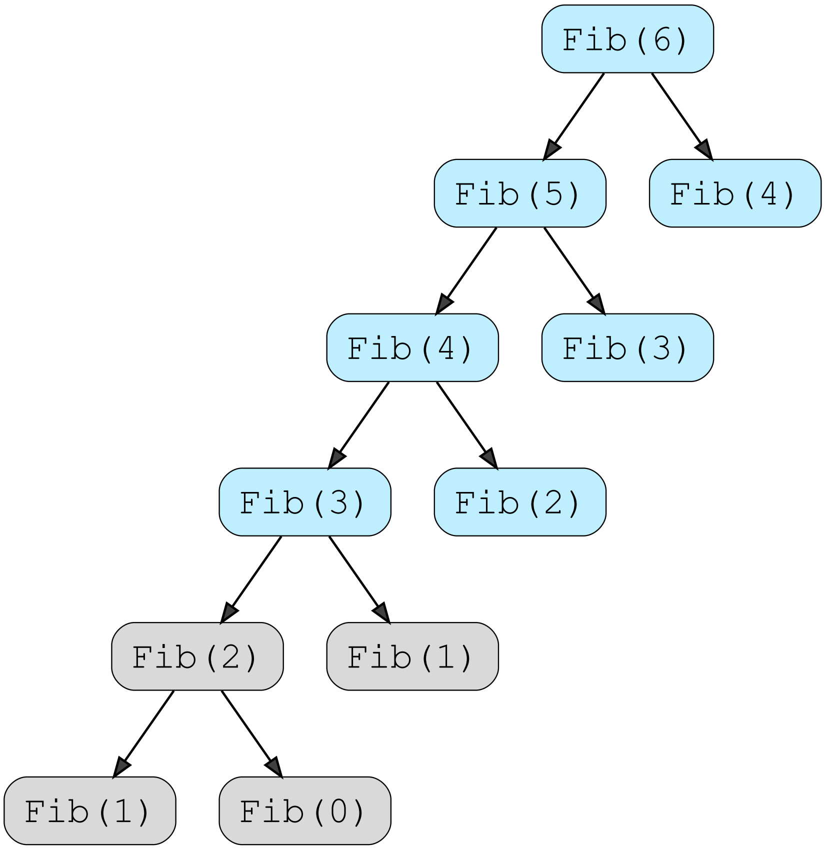
n	result
2	1



base case

Evaluate Fib(6)

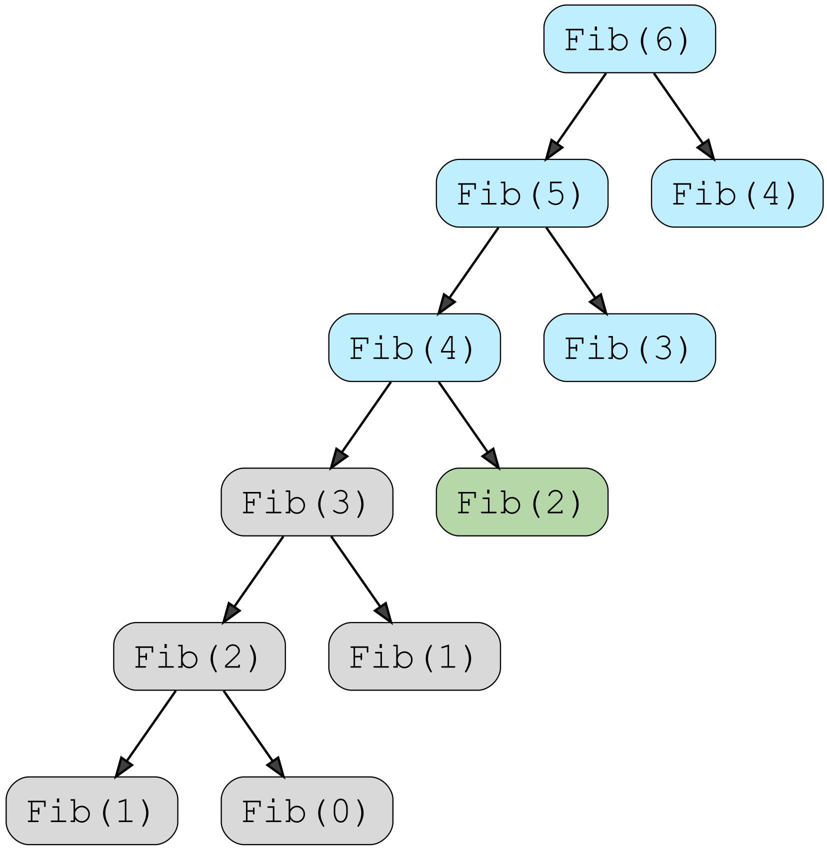
n	result
2	1
3	2



Result evaluated, memoize the result

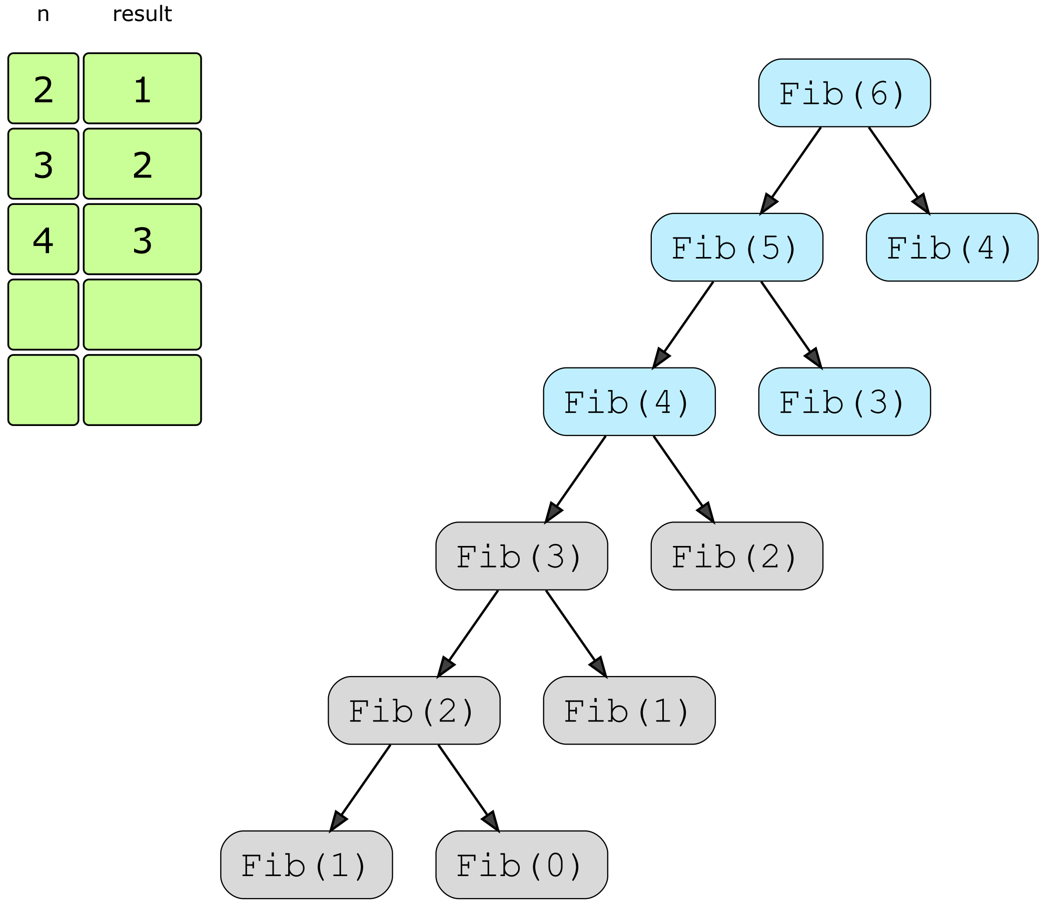
Evaluate Fib(6)

n	result
2	1
3	2



Result available from memoization, use it

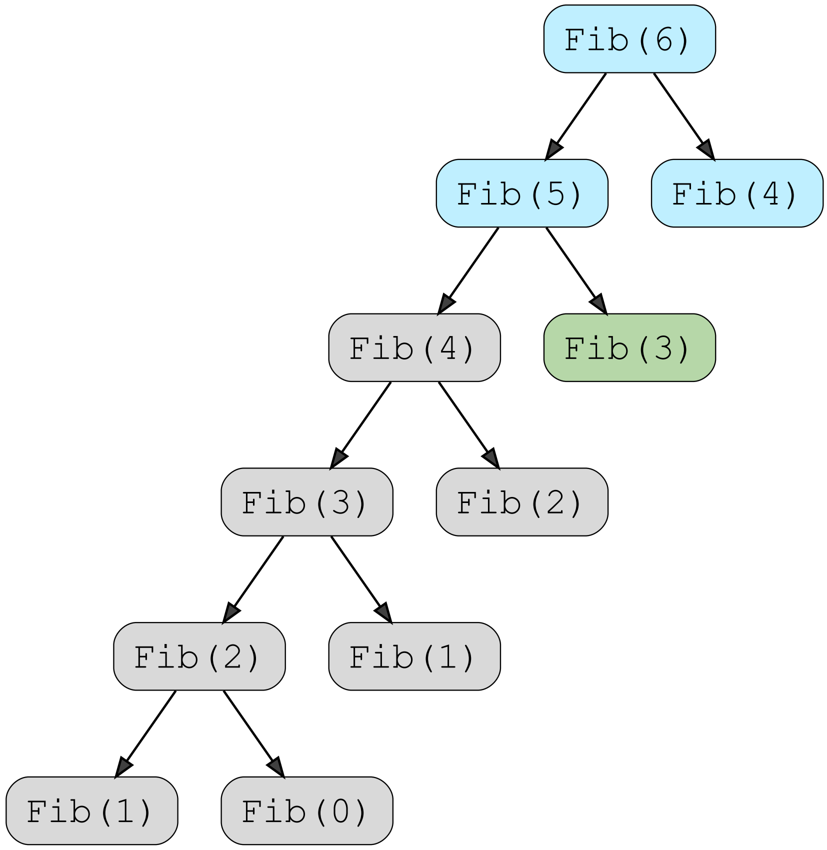
Evaluate Fib(6)



Result evaluated, memoize the result

Evaluate Fib(6)

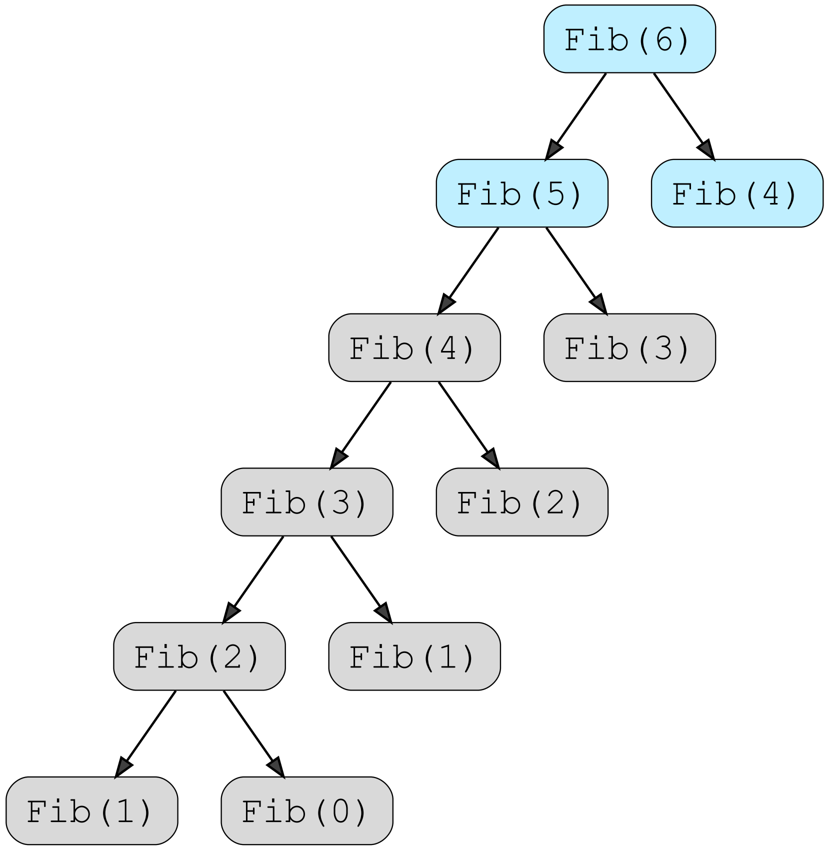
n	result
2	1
3	2
4	3



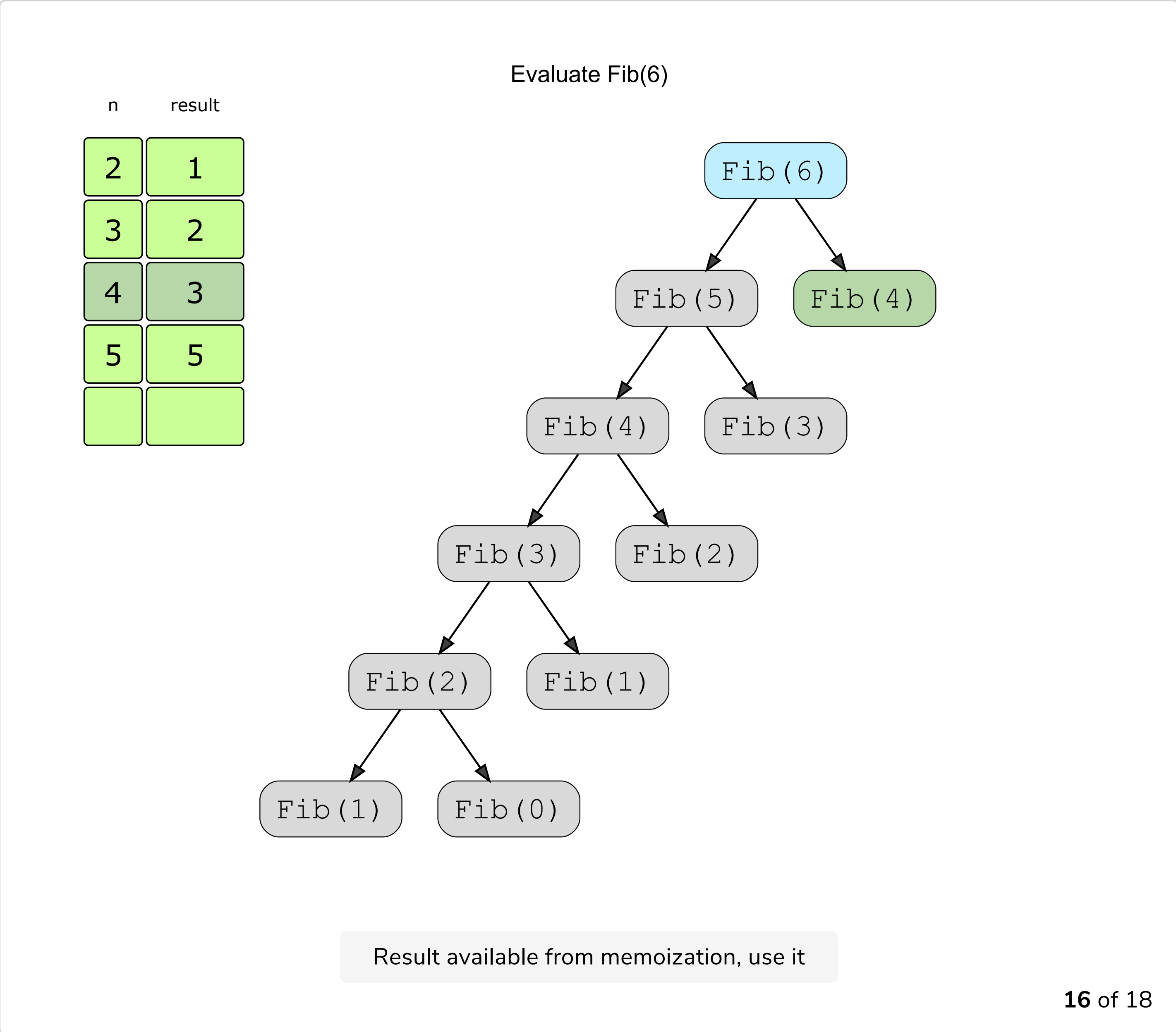
Result available from memoization, use it

Evaluate Fib(6)

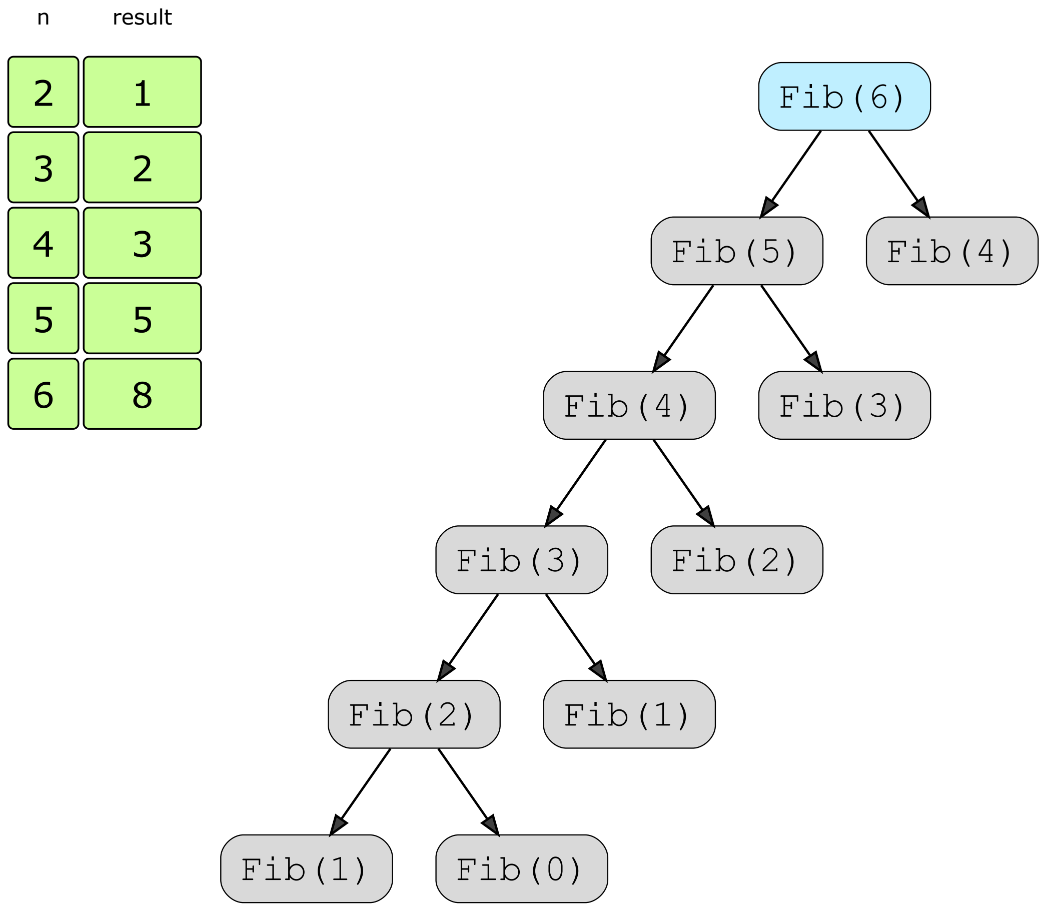
n	result
2	1
3	2
4	3
5	5



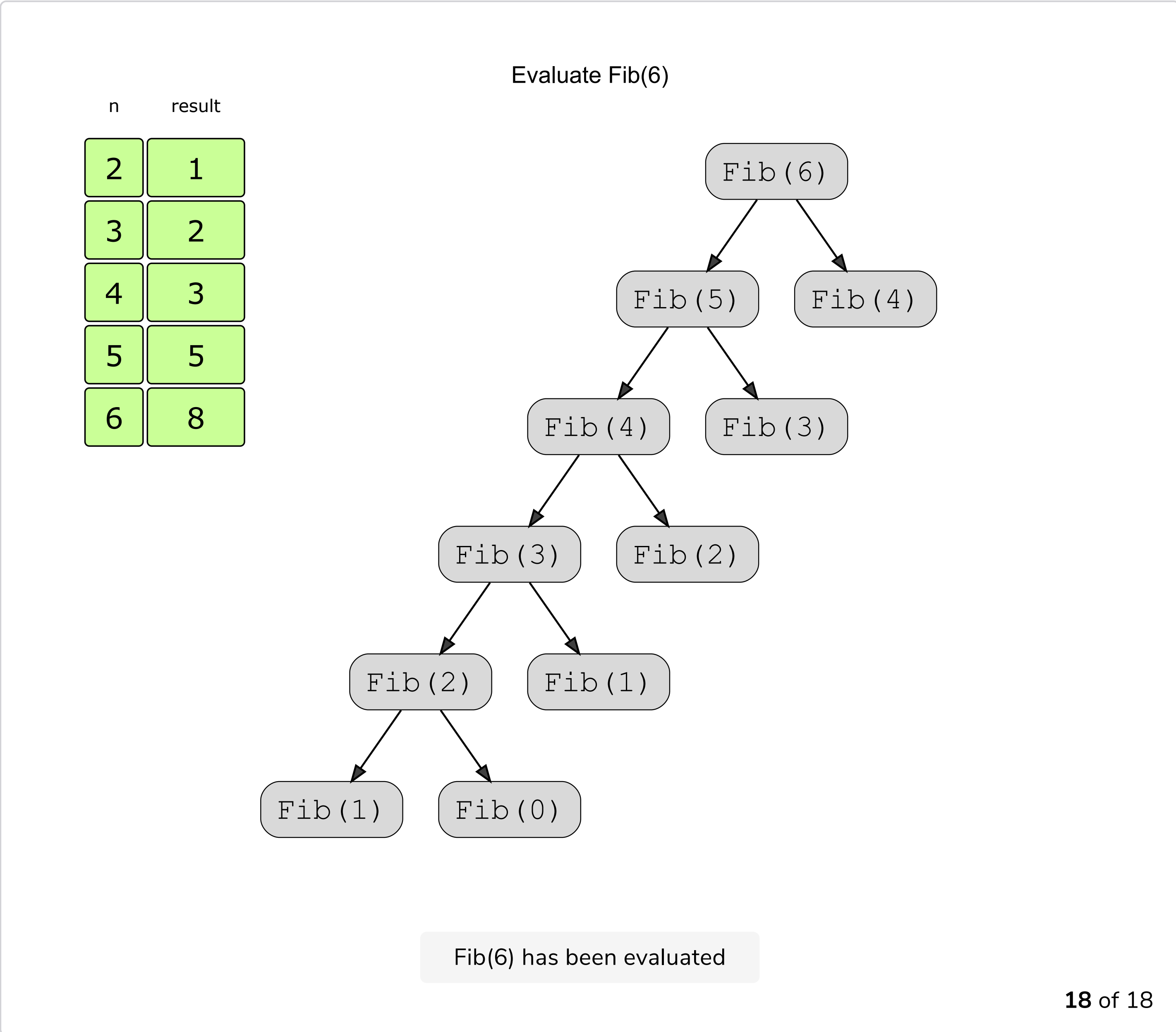
Result evaluated, memoize the result


Evaluate Fib(6)
n
result
2
1
3
2
4
3
5
5
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Result available from memoization, use it
16 of 18

Evaluate Fib(6)



Result evaluated, memoize the result



In the next lesson, you will work on a coding challenge.