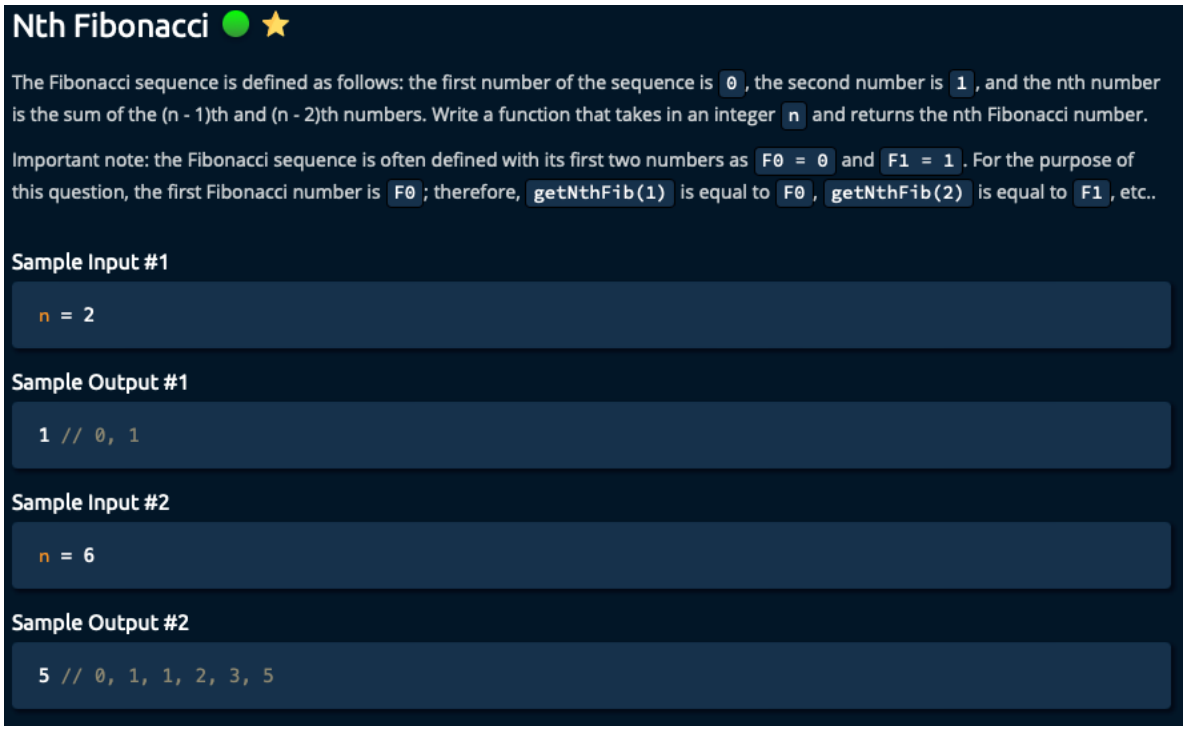
Code Challenge #6 Nth Fibonacci (Easy)



Nth Fibonacci

The Fibonacci sequence is defined as follows: the first number of the sequence is `0`, the second number is `1`, and the nth number is the sum of the (n - 1)th and (n - 2)th numbers. Write a function that takes in an integer `n` and returns the nth Fibonacci number.

Important note: the Fibonacci sequence is often defined with its first two numbers as `F0 = 0` and `F1 = 1`. For the purpose of this question, the first Fibonacci number is `F0`; therefore, `getNthFib(1)` is equal to `F0`, `getNthFib(2)` is equal to `F1`, etc..

Sample Input #1

```
n = 2
```

Sample Output #1

```
1 // 0, 1
```

Sample Input #2

```
n = 6
```

Sample Output #2

```
5 // 0, 1, 1, 2, 3, 5
```

Solution #1

```
1. function getNthFib(n) {
2.   if (n == 2) {
3.         return 1;
4.   } else if (n === 1) {
5.         return 0;
6.   } else {
7.         return getNthFib(n - 1) + getNthFib(n -2);
8.
9.   }
10. }
11.
```

Explanation

The Fibonacci sequence is based on a mathematical concept where each number is the sum of the two previous numbers. This continues on to infinity and a perfect example on how to use a recursive solution to the problem.

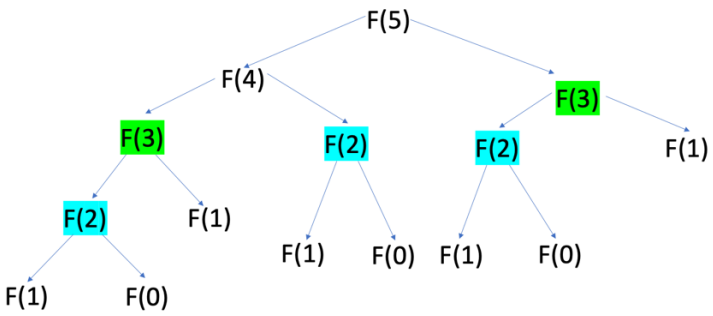
Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987 ...

Each number is the sum of the previous two numbers.

In the coding challenge we are required to find the nth number in the Fibonacci sequence. For example, we said we want to find the sixth number in the Fibonacci sequence we will go through the list and find the 6th number in the list. The 6th number is the number 5 in the Fibonacci sequence. This problem is ideal for recursion because the nth number is found by adding the previous two numbers. If we know the nth value of the previous two numbers we can find the nth value for the value we are looking for. In the function we need to define base cases where we terminate the recursive call. The two base cases will be if n = 2 we will return one or if n = 1 we will return 0 since the second number is equal to 1in Fibonacci sequence and zero for the first number in Fibonacci sequence. If those conditions are not met, we will return the recursion of n – 1 plus recursion of n – 2. The runtime for this will be 2^n because each call on the recursion function (approximately) doubles the previous number of calls. This is highly inefficient and solution two will address a better (less memory) solution.

Recursive Function Calls for Fibonacci of 5



Solution #2

```
1. function getNthFib(n, memoize = {1: 0, 2: 1}) {
2.   if (n in memoize) {
3.         return memoize [n];
4.   } else {
5.         memoize[n] = getNthFib(n - 1, memoize) + getNthFib(n - 2,
   memoize);
6.         return memoize[n]
7.   }
8. }
```

Explanation

The second solution uses an object called memoize which stores the nth number as a value in a key value pair. The main function getNthFib takes in two arguments which are n and memoize. Memoize starts off with a key value par of 1: 0, 2:1. If n is found in memoize we return memoize[n] else we find memoize[n] by calling getNthFib(n – 1, memoize) + getNthFib(n – 2, memoize). Once we calculate memoize we return memoize[n]. This is solution is O(n) because we don't repeat calculates several times like in solution 1. Each previous solution in the recursive call stack is stored in an object for future use. This helps to lower the runtime from O(2^n) to O(n).

Solution #3

```
1. function getNthFib(n) {
2.   const lastTwo = [0, 1];
3.  let counter  = 3;
4.  while (counter <= n){
5.         const nextFib = lastTwo[0] + lastTwo[1];
6.         lastTwo[0] = lastTwo[1];
7.         lastTwo[1] = nextFib;
8.         counter++;
9.  }
10.        return n > 1 ? lastTwo[1] : lastTwo[0]
11. }
12.
```

Explanation

This explanation requires that we create an array with two starting initial values of 0 and 1. The array is called const lastTwo because it keeps track of the last two values in order to get the next Fibonacci number. We also start with a let counter with an initial value of 3. We start with 3 because lastTwo is given in the first two values of the Fibonacci sequence. Any value 3 or greater will require us to calculate the Fibonacci values in the while loop. We then initiate an while loop which runs as long as the counter is less than or equal to the nth value. If it meets those conditions then the while loop will continue to run. In the while loop we will calculate the nextFib number by adding the lastTwo[0] + lastTwo[1]. We then assign the value of lastTwo[1] to lastTwo[0]. We then assign the value of nextFib to lastTwo[1]. The final part of the while loop is to increment the counter in order to keep calculating the nth Fibonacci number. Once we are done with the while loop we will go to the return statement. The return statement is a ternary statement which says that if n > 1 we will return lastTwo[1] else it will return lastTwo[2]. We use greater than 1 because if it is one we will return lastTwo[0] instead of lastTwo[1] due to how ternary statement is structured. The code runs in O(n) time.