파이썬으로 배우는 데이터 구조

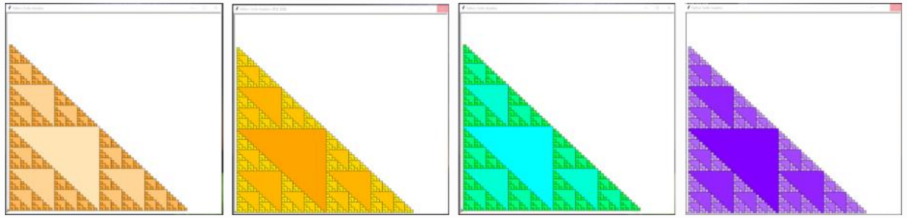
한동대학교
전산전자공학부
김영섭 교수

학습 목표

재귀함수를 구현하는 과정을 stack을 통해 이해하고
memoization을 활용할 수 있다

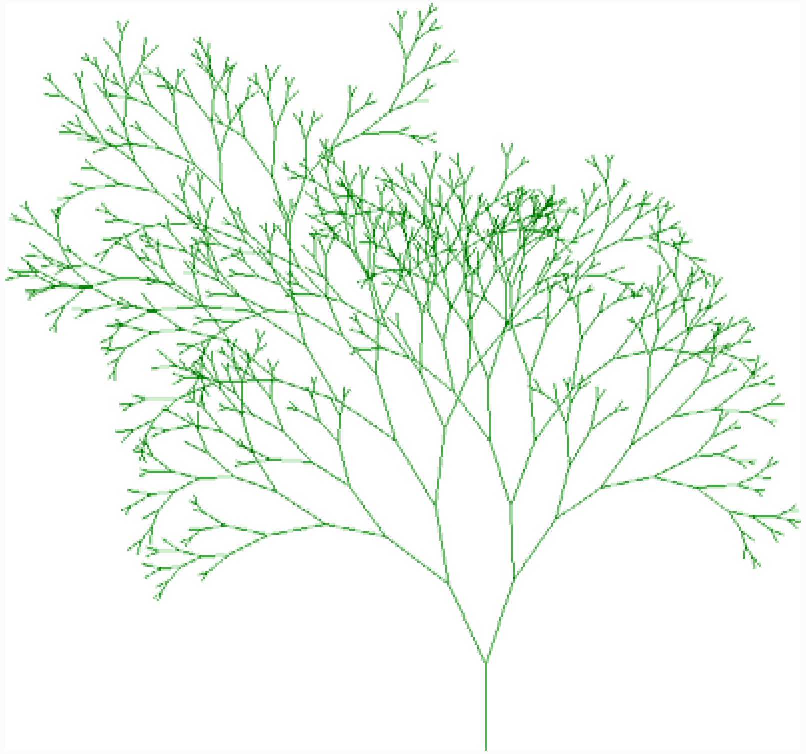
Data Structures in Python
Chapter 4

- Recursion Concepts
- **Recursion Stack and Memoization**
- Recursive Algorithms
- Recursive Graphics
- Exercise - Stacking boxes



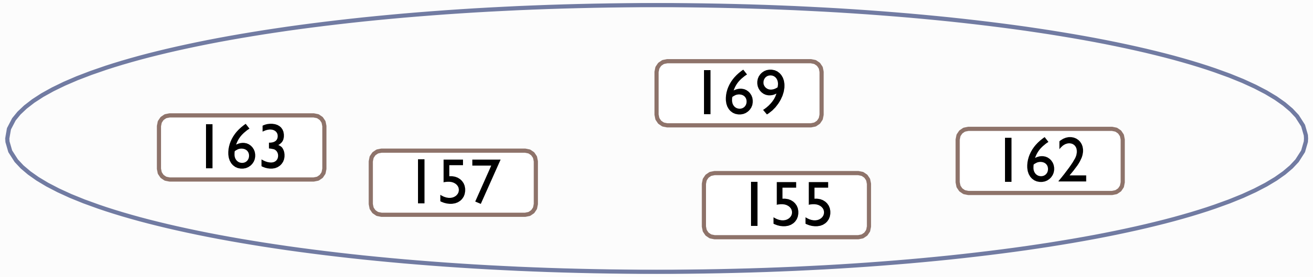
Agenda

- Recursion and Stack
  - The Fibonacci Sequence
  - Using Memoization



Recursion and Stack

- Example: Find the tallest person in a group of N > 0 students



Who is the tallest?

```
def find_tallest(Group_of_students)
    tallest = Take any student from group;
    Repeat until nobody left
        Take next student from group
        If student is taller than tallest then
            tallest = student
    Return tallest
```

Iterative solution

Recursion and Stack

- Example: Find the tallest person in a group of N > 0 students

```
[162, 155, 169, 157]
```

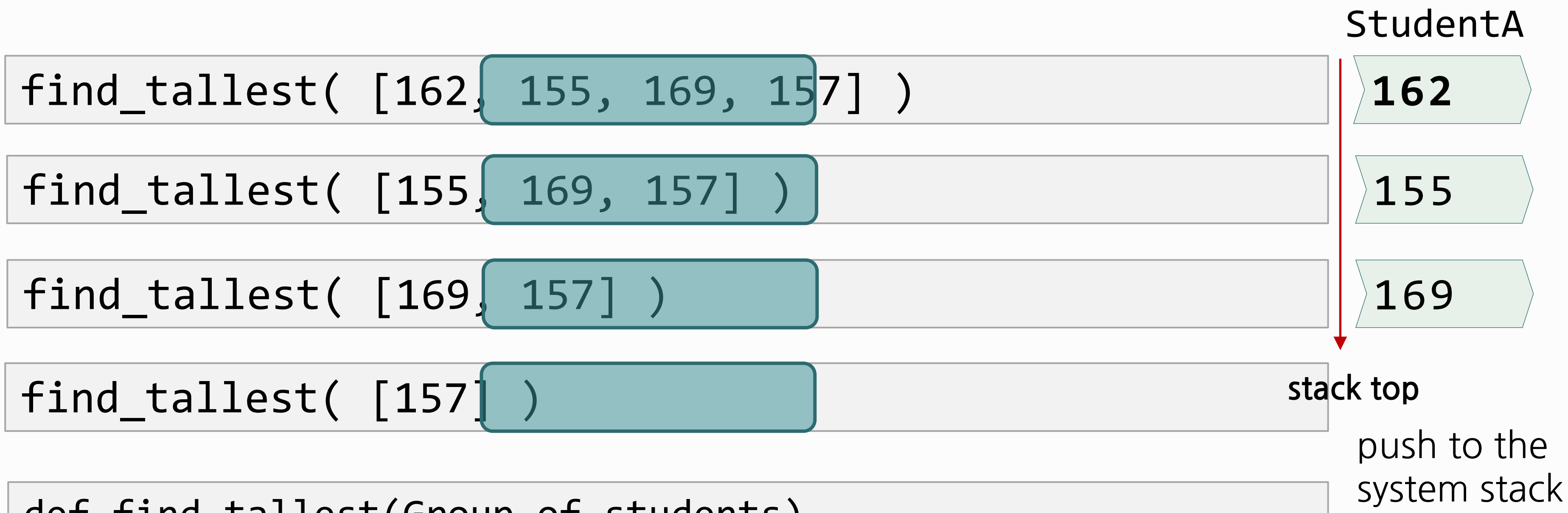
```
def find_tallest(students):
    if len(students) == 1:
        return students[0]

    a = students[0]
    b = find_tallest(students[1:])
    return a if a > b else b
```

```
def find_tallest(Group of students)
    If only one student in group
        return this student
    else
        StudentA = Take one student from group
        StudentB = find_tallest(Remaining Group)
    return the taller person of StudentA and StudentB
```

Recursion and Stack

- Example: Find the tallest person in a group of N > 0 students



StudentA

```
find_tallest( [162, 155, 169, 157] )
find_tallest( [155, 169, 157] )
find_tallest( [169, 157] )
find_tallest( [157] )
```

162

155

169

stack top

push to the system stack

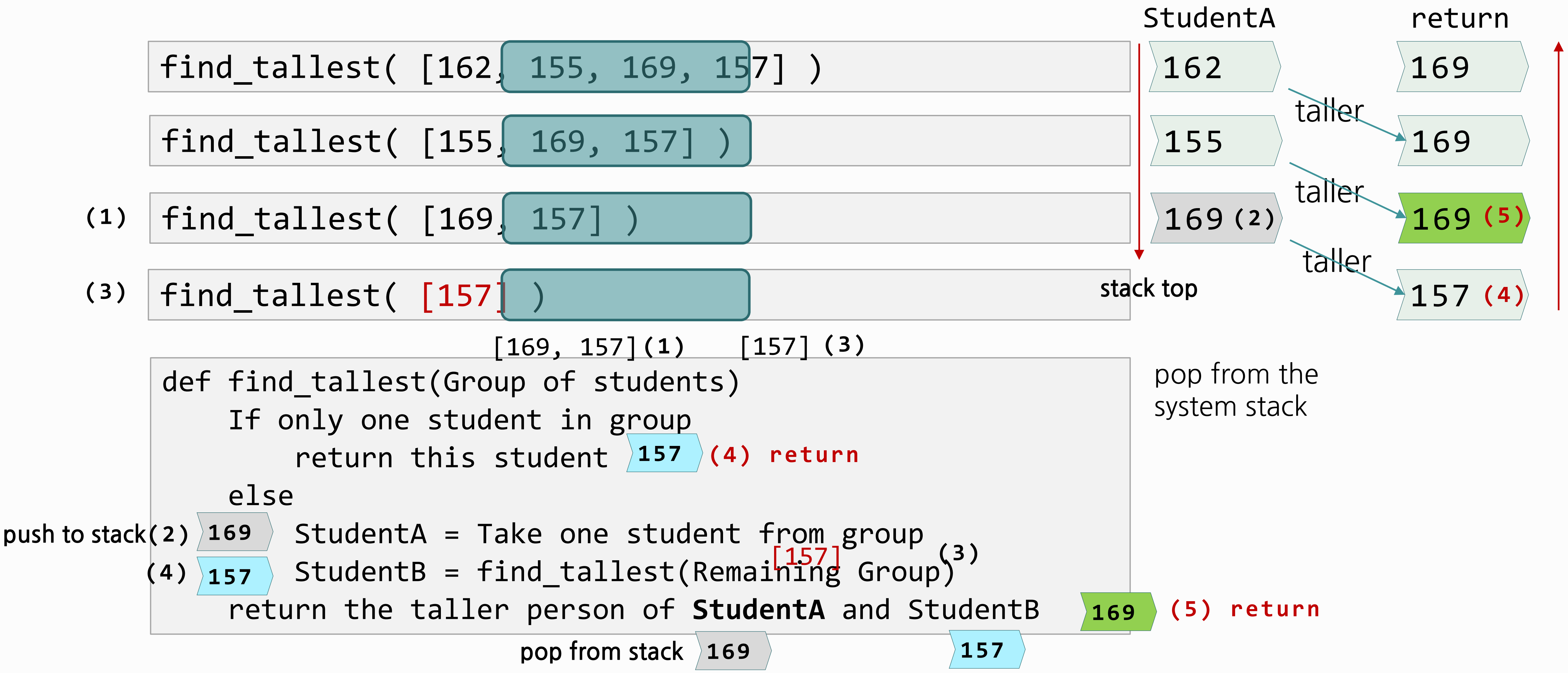
```
def find_tallest(Group of students)
    If only one student in group
        return this student
    else
        StudentA = Take one student from group
        StudentB = find_tallest(Remaining Group)
    return the taller person of StudentA and StudentB
```

162 push to stack

155, 169, 157

Recursion and Stack

- Example: Find the tallest person in a group of N > 0 students



StudentA

return

```
    find_tallest( [162, 155, 169, 157] )
    find_tallest( [155, 169, 157] )
(1) find_tallest( [169, 157] )
(3) find_tallest( [157] )
```

StudentA		return
162		169
155	taller	169
169 (2)	taller	169 (5)
stack top	taller	157 (4)

[169, 157] (1)   [157] (3)

```
def find_tallest(Group of students)
    If only one student in group
        return this student
    else
        StudentA = Take one student from group
        StudentB = find_tallest(Remaining Group)
    return the taller person of StudentA and StudentB
```

157 (4) return

push to stack (2) 169

(4) 157

[157] (3)

169 (5) return

pop from stack 169   157

pop from the system stack

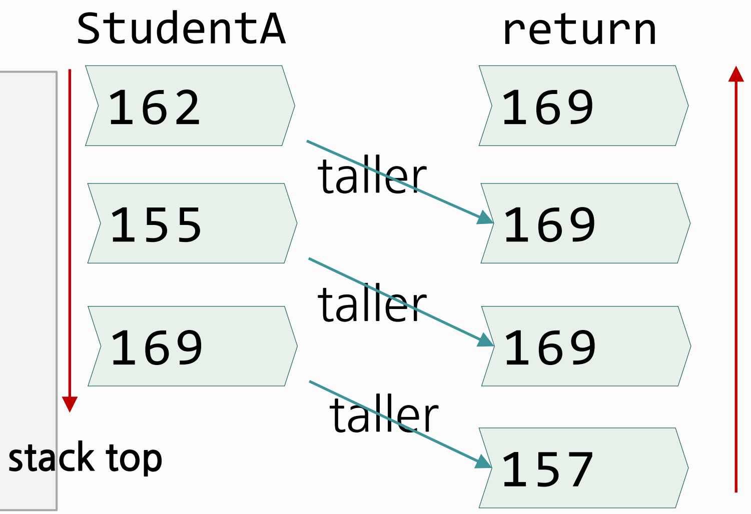
Recursion and Stack

- Example: Find the tallest person in a group of N > 0 students

`[162, 155, 169, 157]`

```
def find_tallest(students):
    if len(students) == 1:
        return students[0]

    a = students[0]
    b = find_tallest(students[1:])
    return a if a > b else b
```

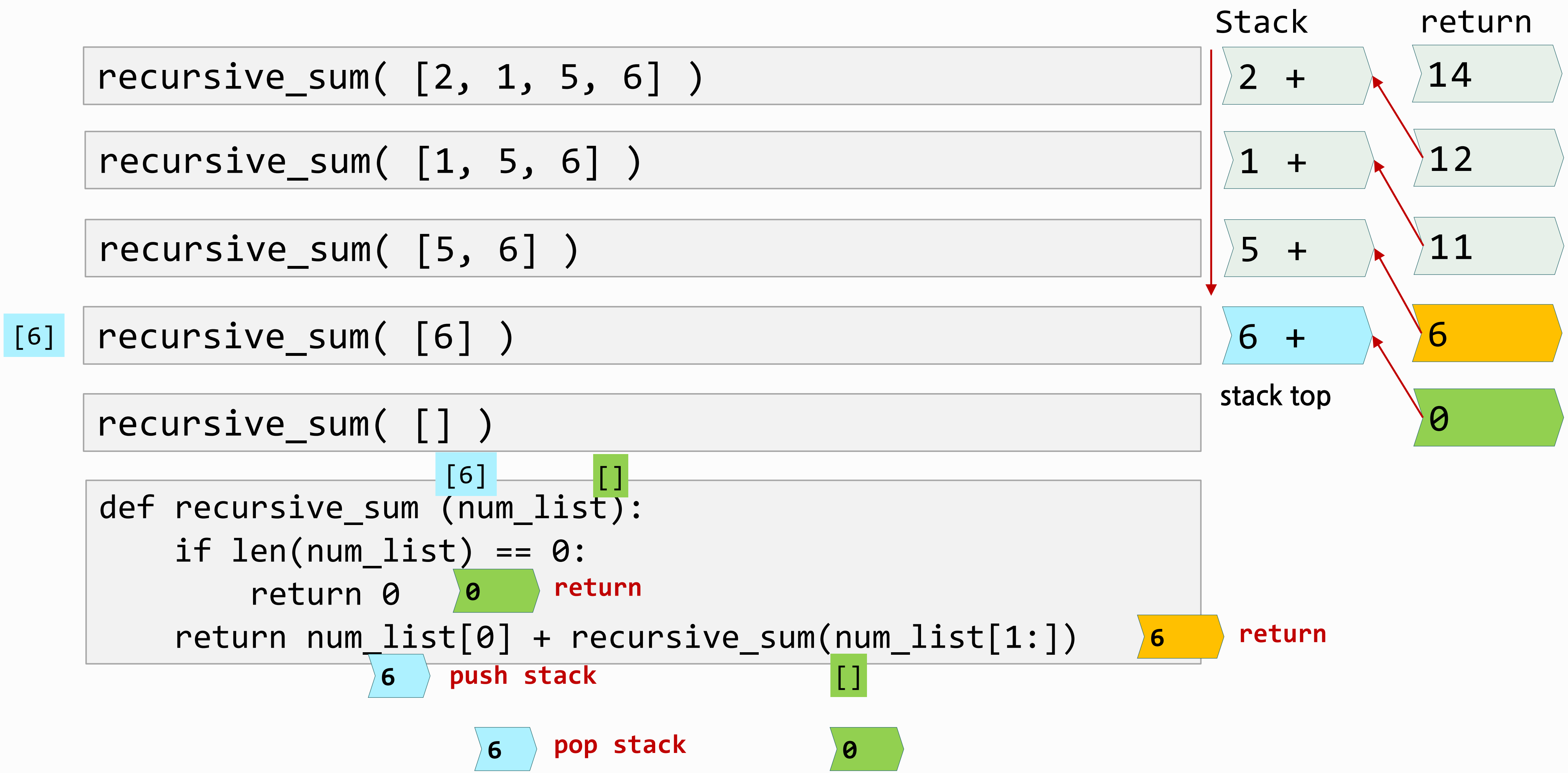


pop from the system stack

```
def find_tallest(Group of students)
    If only one student in group
        return this student
    else
        StudentA = Take one student from group
        StudentB = find_tallest(Remaining Group)
    return the taller person of StudentA and StudentB
```

Exercise: Recursion and Stack

- Get the recursive sum by taking the first number + the sum of the rest of the list.



Stack

return

recursive_sum([2, 1, 5, 6])

2 +

14

recursive_sum([1, 5, 6])

1 +

12

recursive_sum([5, 6])

5 +

11

[6]

recursive_sum([6])

6 +

6

stack top

recursive_sum([])

0

[6]

[]

```
def recursive_sum (num_list):
    if len(num_list) == 0:
        return 0
    return num_list[0] + recursive_sum(num_list[1:])
```

0 return

6 return

6 push stack

[]

6 pop stack

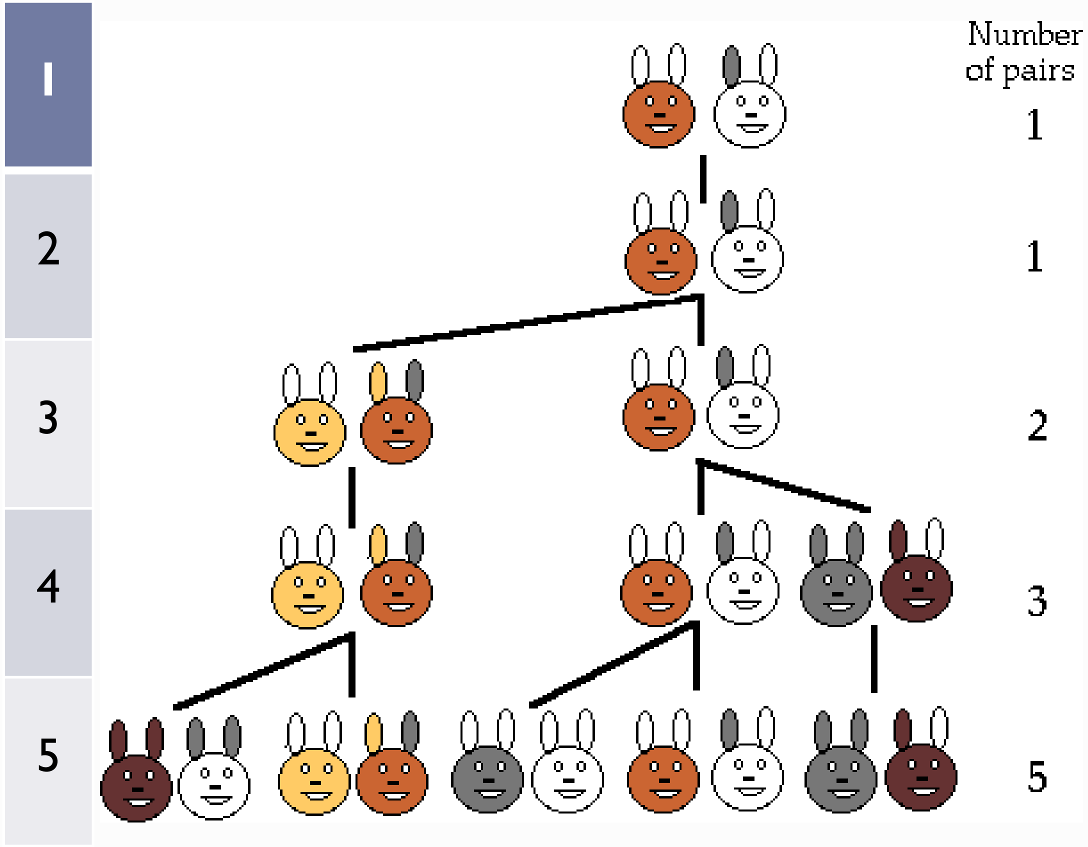
0

The Fibonacci Sequence

- Describes the growth of an idealized (biologically unrealistic) rabbit population, assuming that:
  - Rabbits never die.
  - A rabbit reaches sexual maturity exactly two months after birth, that is, at the beginning of its third month of life.
  - Rabbits are always born in male-female pairs.
  - At the **beginning** of every month, each sexually mature male- female pair gives **birth** to exactly one male-female pair.

The Fibonacci Sequence

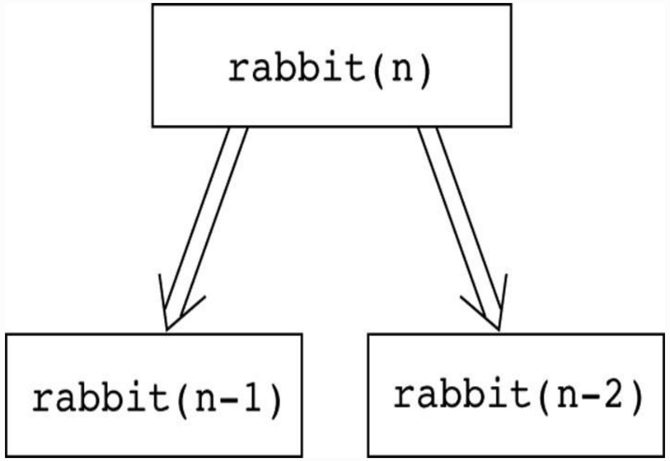
- Problem:
  - How many pairs of rabbits are alive in month n?
- Example:
  - rabbit(5) = 5



- Recurrence relation
  - rabbit(n) = rabbit(n-1) + rabbit(n-2)

The Fibonacci Sequence - Recursive Definition

- Base cases
  - rabbit(2), rabbit(1)
- Recursive case
  - $$\text{rabbit}(n) = \begin{cases} 1 \text{ if } n \text{ is } 1 \text{ or } 2 \\ \text{rabbit}(n-1) + \text{rabbit}(n-2) \text{ if } n > 2 \end{cases}$$



- Fibonacci sequence
  - The series of numbers fibo(1), fibo(2), fibo(3), and so on
  - The sequence of numbers fibo(n) for all n is called **Fibonacci Sequence** or **Fibonacci numbers.**

```
def fibo(n):
    """Assume n >= 0 """
    if n < 2:
        return 1
    return fibo(n-1) + fibo(n-2)
```

The Fibonacci Sequence - Coding Exercise

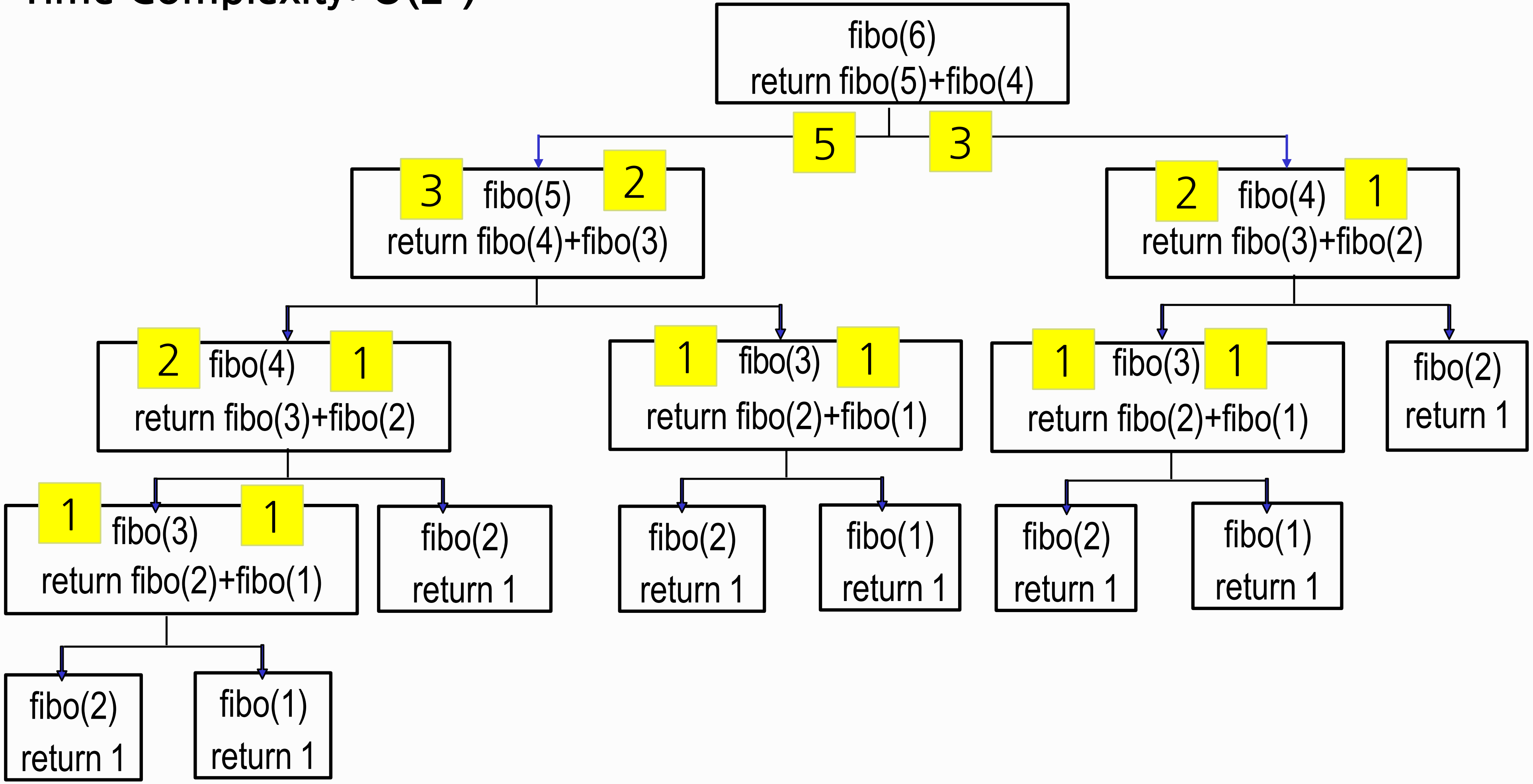
- Rewrite the fibo() using a ternary operator to replace 'None'.

```
def fibo(n):
    """Assume n >= 0 """
    return None
```

```
def fibo(n):
    """Assume n >= 0 """
    if n < 2:
        return 1
    return fibo(n-1) + fibo(n-2)
```

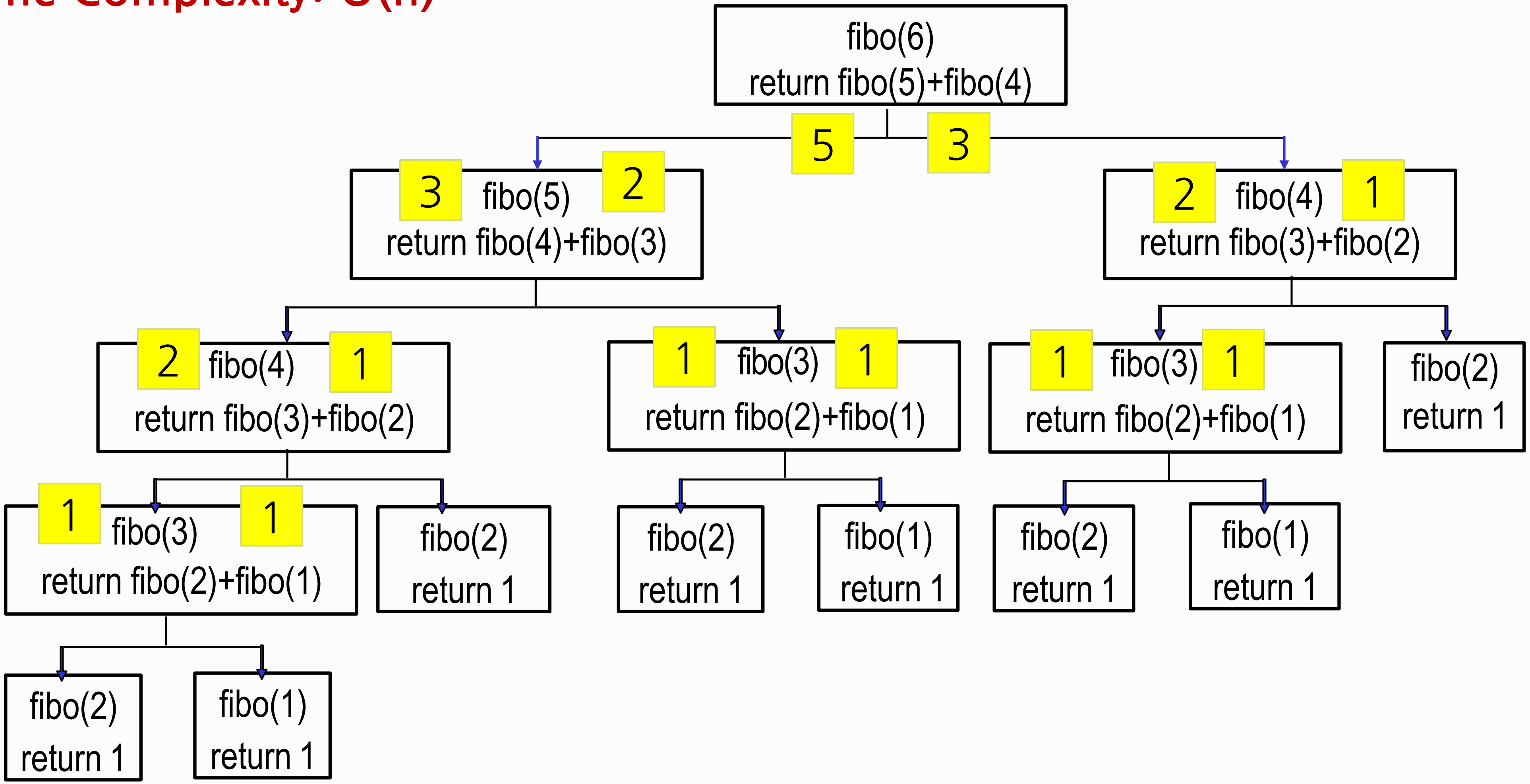
The Fibonacci Sequence - Examples

- fibo(6) = 8
  - How many times were fibo(2) and fibo(3) called to compute fibo(6), respectively?
  - **Time Complexity: O(2^n)**



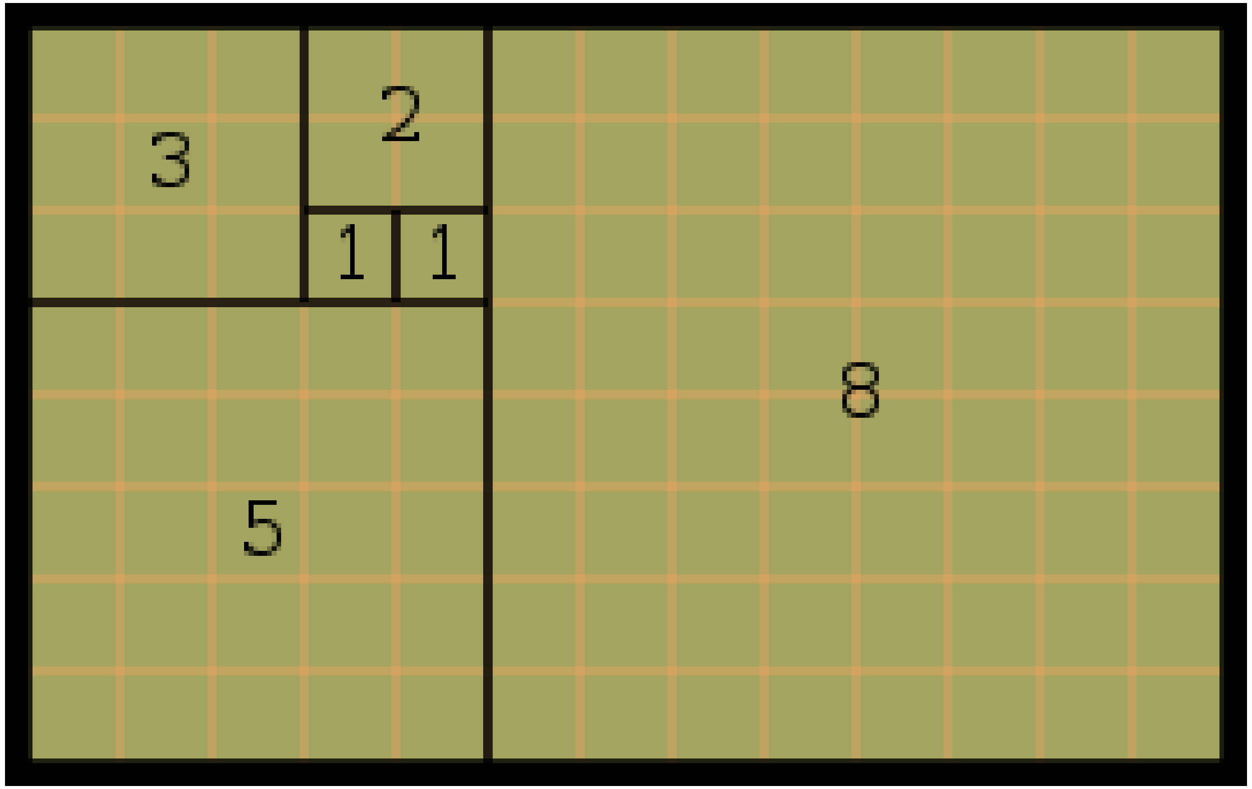
The Fibonacci Sequence - Examples

- Rather computing the same terms repeatedly, just save them in a set and reuse them whenever necessary. This technique is called **memoization, not memorization**.
- **Time Complexity: O(n)**



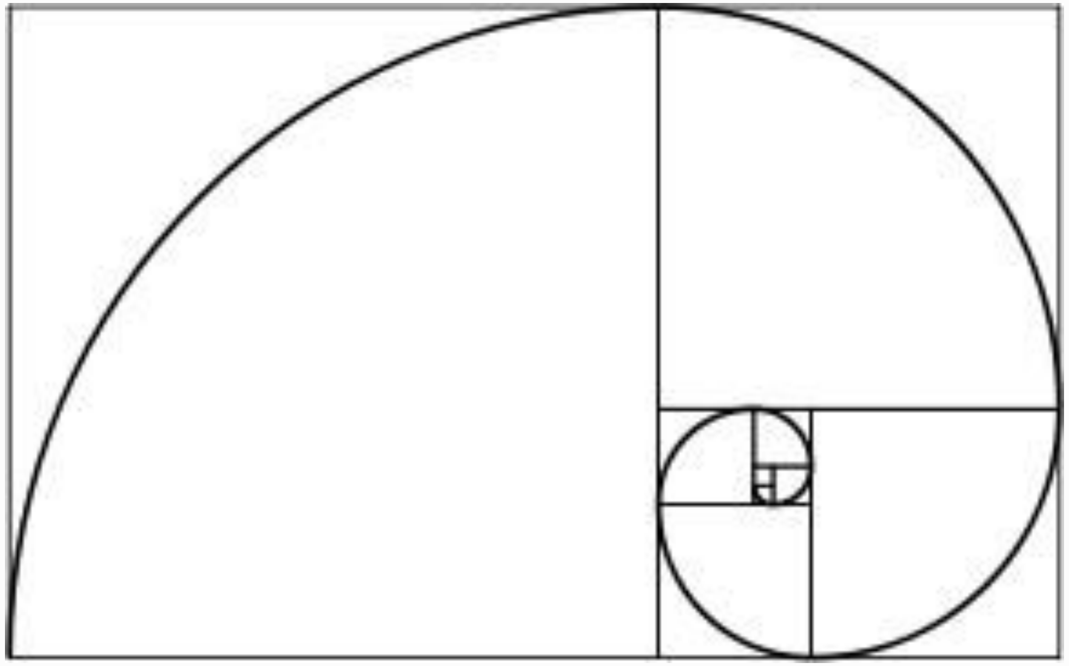
The Fibonacci Sequence - Examples

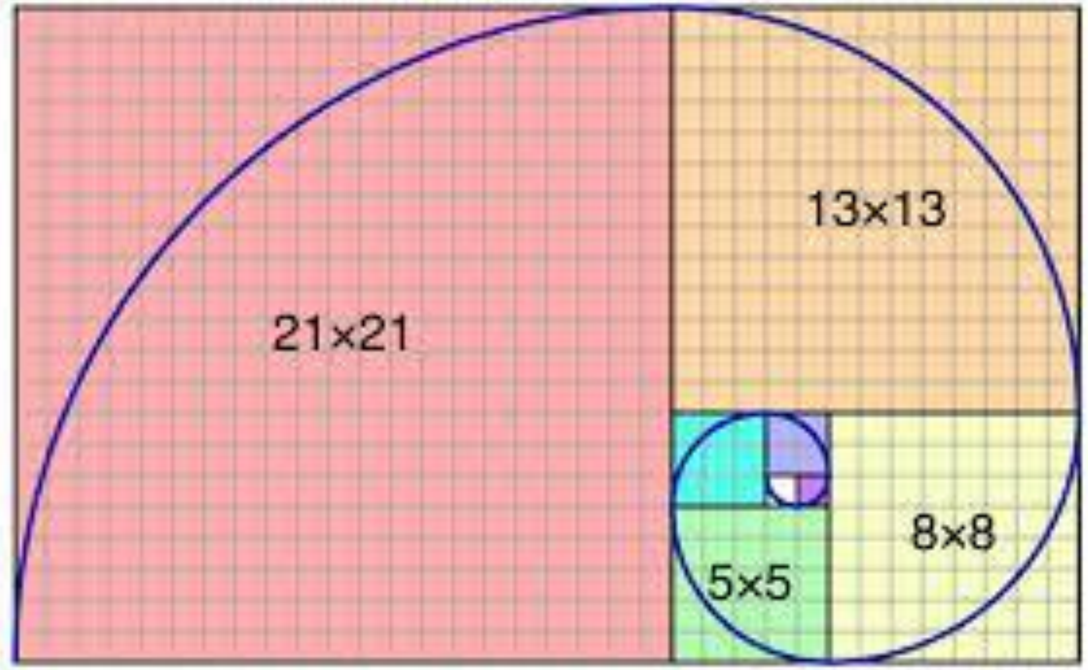
- Fibonacci Tiling



The Fibonacci Sequence - Examples

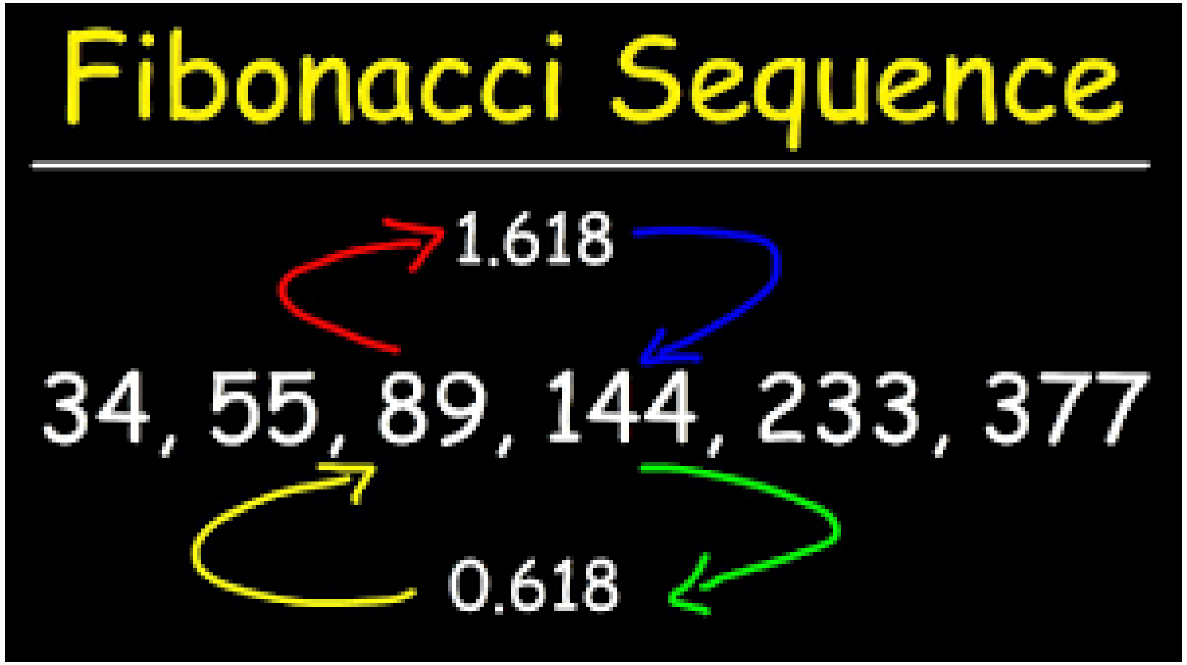
- Fibonacci Spiral

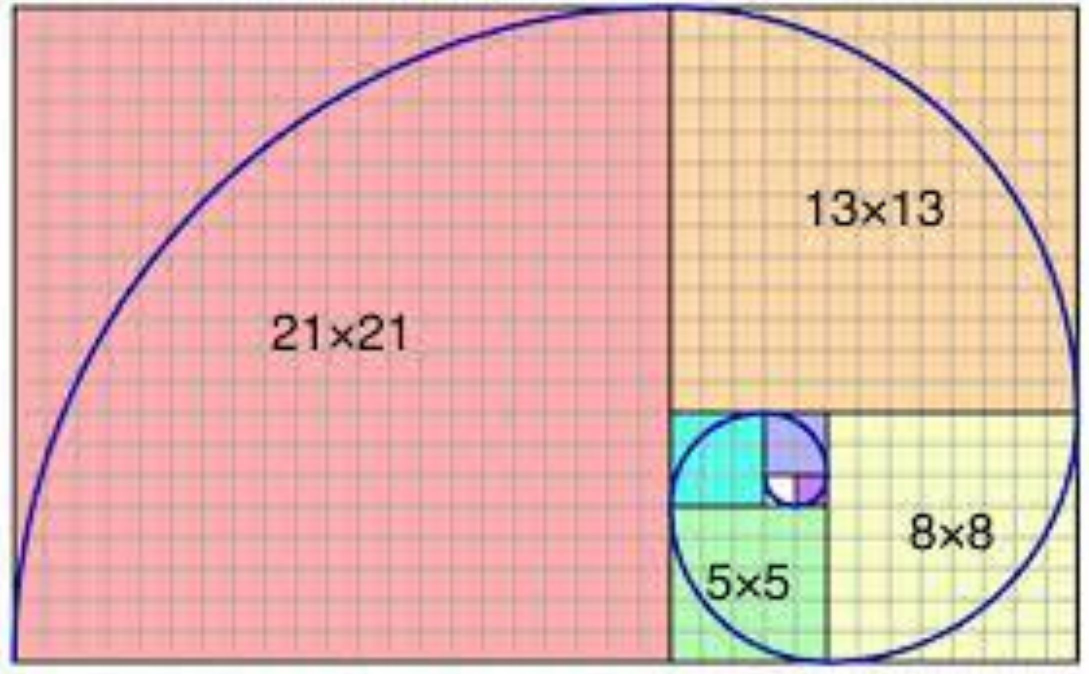




The Fibonacci Sequence - Examples

- Fibonacci and the Golden Ratio - https://www.youtube.com/watch?v=mVO2dcuR7P0





The Fibonacci Sequence - Coding Exercise

- Use **math module** in Python to compute fibo(n) and print the output as shown. Print n and numbers are right-justified and fibo(n) results are left-justified.
- Refer to here for the fibo(n) formular.

```
  n fibo(n)
  0 0
 10 55
 20 6765
 30 832040
 40 102334155
 50 12586269025
 60 1548008755920
 70 190392490709135
 80 23416728348467744
 90 2880067194370824704
100 354224848179263111168
110 43566776258855008468992
120 5358359254990987687100416
130 659034621587632984143429632
140 81055900096023879930404143104
150 9969216677189352939733964029952
160 1226132595394194733041959223427072
170 150804340016808806258572755667517440
180 18547707689472097530613662299140915200
190 2281217241465051689432021623822983626752
200 280571172992512015699912586503521287798784
```

```
import math

def fibo(n):
    return None

if __name__=='__main__':
    print(None)
    for n in range(0, 201, 10):
        fibo(n)
        print(None)
```

The Fibonacci Sequence - Coding Exercise

- **Idea:** Rather computing the same terms repeatedly, save them in a in a dictionary and reuse them whenever necessary. This technique is called **memoization**
  - For example, fibo_memo has the following elements when n = 0 ~ 11.
    `{ 0: 0, 1: 1, 2: 1, 3: 2, 4: 3, 5: 5, 6: 8, 7: 13, 8: 21, 9: 34, 10: 55 }`
- Rewrite fibo() using a memoization and a ternary operator to replace 'None'.

```
def fibo(n):
    """Assume n >= 0 """
    if n < 2:
        return 1
    return fibo(n-1) + fibo(n-2)
```

```
fibo_memo = {}
def fibo(n):
    if n not in fibo_memo:
        None
    return None
```

The Fibonacci Sequence - Coding Exercise

- Make the following code complete by replacing the 'None' to reproduce the output shown. Notice that n and numbers are right-justified and fibo(n) results are left-justified.

```
  n fibo(n)
  0 0
 10 55
 20 6765
 30 832040
 40 102334155
 50 12586269025
 60 1548008755920
 70 190392490709135
 80 23416728348467744
 90 2880067194370824704
100 354224848179263111168
110 43566776258855008468992
120 5358359254990987687100416
130 659034621587632984143429632
140 81055900096023879930404143104
150 9969216677189352939733964029952
160 1226132595394194733041959223427072
170 150804340016808806258572755667517440
180 18547707689472097530613662299140915200
190 2281217241465051689432021623822983626752
200 280571172992512015699912586503521287798784
```

```
fibo_memo = {}
def fibo(n):
    if n not in fibo_memo:
        None
    return fibo_memo[n]

if __name__=='__main__':
    print(None)
    for n in range(0, 201, 10):
        fibo(n)
        print(None)
```

The Fibonacci Sequence - Coding Exercise

- Modify the following code such that it does not use the global variable fibo_memo.

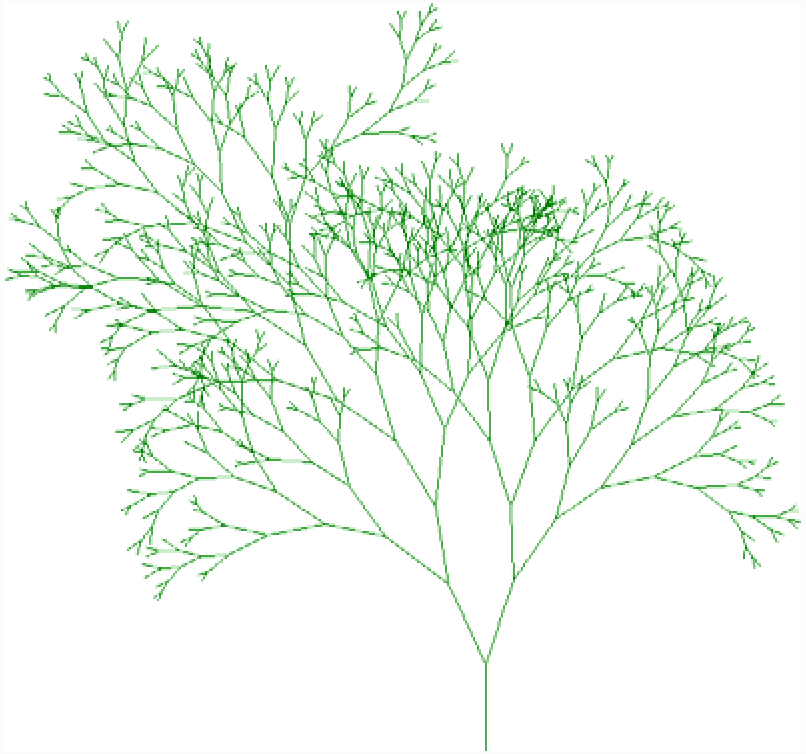
```
fibo_memo = {}
def fibo(n):
    if n not in fibo_memo:
        None
    return fibo_memo[n]

if __name__=='__main__':
    print(None)
    for n in range(0, 201, 10):
        fibo(n, fibo_memo)
        print(None)
```

```
  n fibo(n)
  0 0
 10 55
 20 6765
 30 832040
 40 102334155
 50 12586269025
 60 1548008755920
 70 190392490709135
 80 23416728348467744
 90 2880067194370824704
100 354224848179263111168
110 43566776258855008468992
120 5358359254990987687100416
130 659034621587632984143429632
140 81055900096023879930404143104
150 9969216677189352939733964029952
160 1226132595394194733041959223427072
170 150804340016808806258572755667517440
180 18547707689472097530613662299140915200
190 2281217241465051689432021623822983626752
200 280571172992512015699912586503521287798784
```

Summary

- Recursion uses the system stack.
- We may use the memoization to speed up the recursive calls in some cases.



학습 정리

1) 재귀함수는 stack을 이용한다

2) Memoization은 파이썬 딕셔너리를 이용해 값을 저장하여 다시 사용할 수 있게 만든다

3) Memoization을 활용하면 컴퓨터가 같은 작업하는 것을 방지하여 재귀를 수행하는 속도를 높인다

파이썬으로 배우는
데이터 구조

수고했습니다
곧 다음 시간에
다시 뵙겠습니다