

CSC 111:

Intro to Computer Science through Programming

Spring 2017
Prof. Sara Mathieson



Admin

- + Encouraged: Piazza for final review (no more final project help)
- + **Final project** extended til **Thursday at 5pm**
- + TA review session **tonight**: 7:30-9:30pm
- + Practice final during lab today/tomorrow
- + Self-scheduled **final exam** (similar style to the midterm)

Practice Final

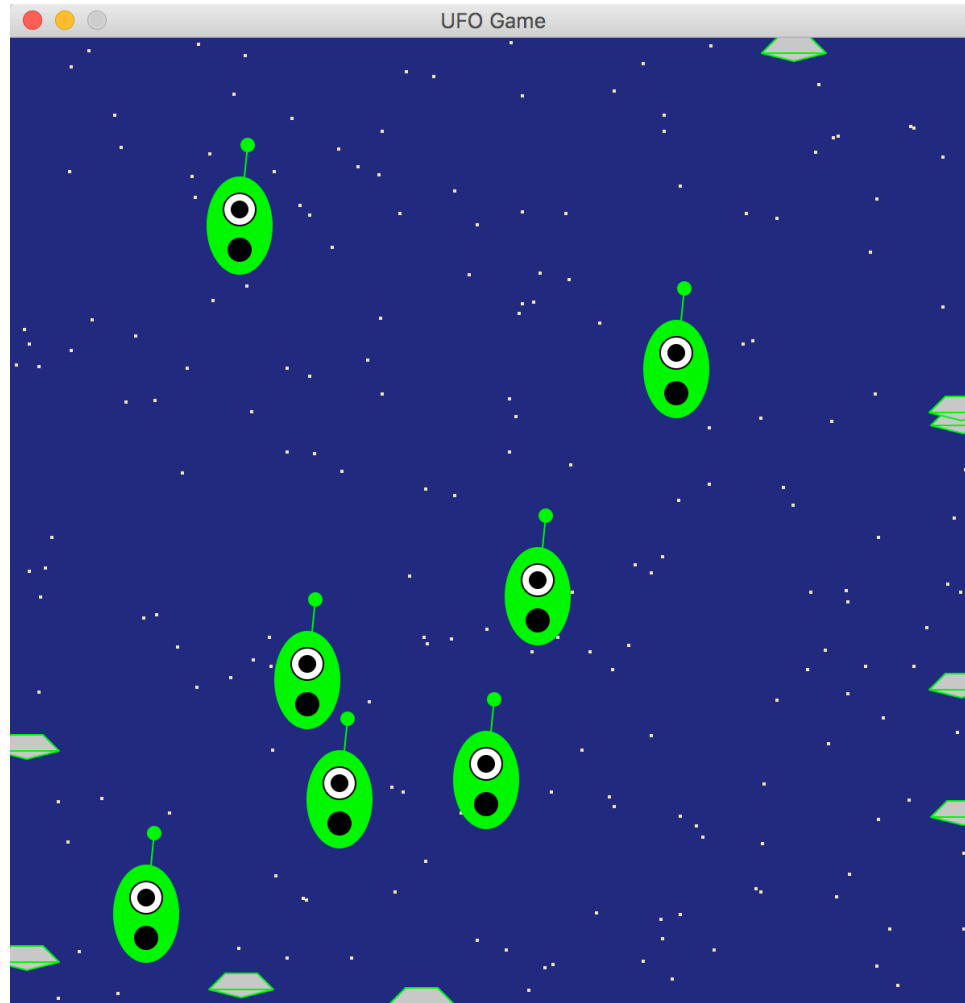
- + During each lab section (you don't have to go to your assigned section)
- + Just like the final: no technology, notes, etc (just 2 cheat sheets)
- + Feel free to write on the backs, but no extra pages
- + TAs will provide feedback
- + I will add **3 points to your final exam score** if you take the practice final during lab, give an honest effort, put your name on it, turn it in, and **PICK IT UP** after feedback has been given

Outline: 5/3

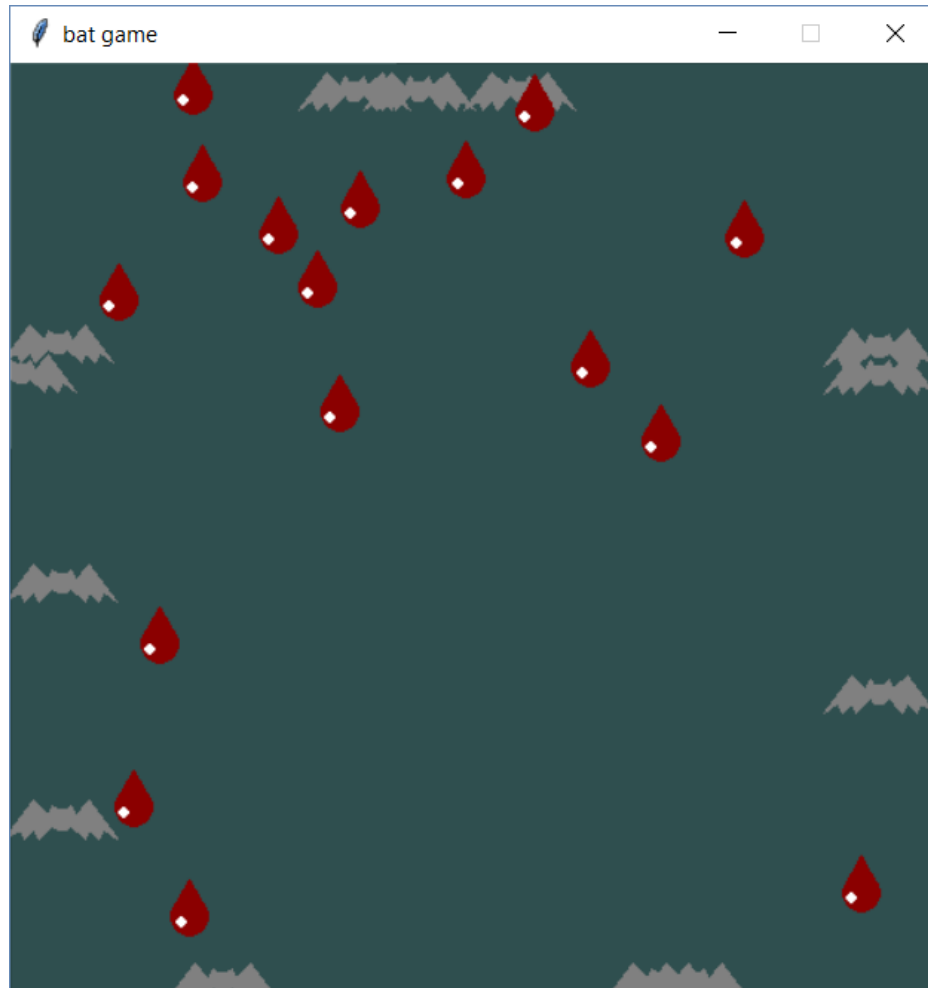
- + Homework demos
- + Finish 99 number activity
- + Review **recursion**
- + Review **classes**

Homework examples

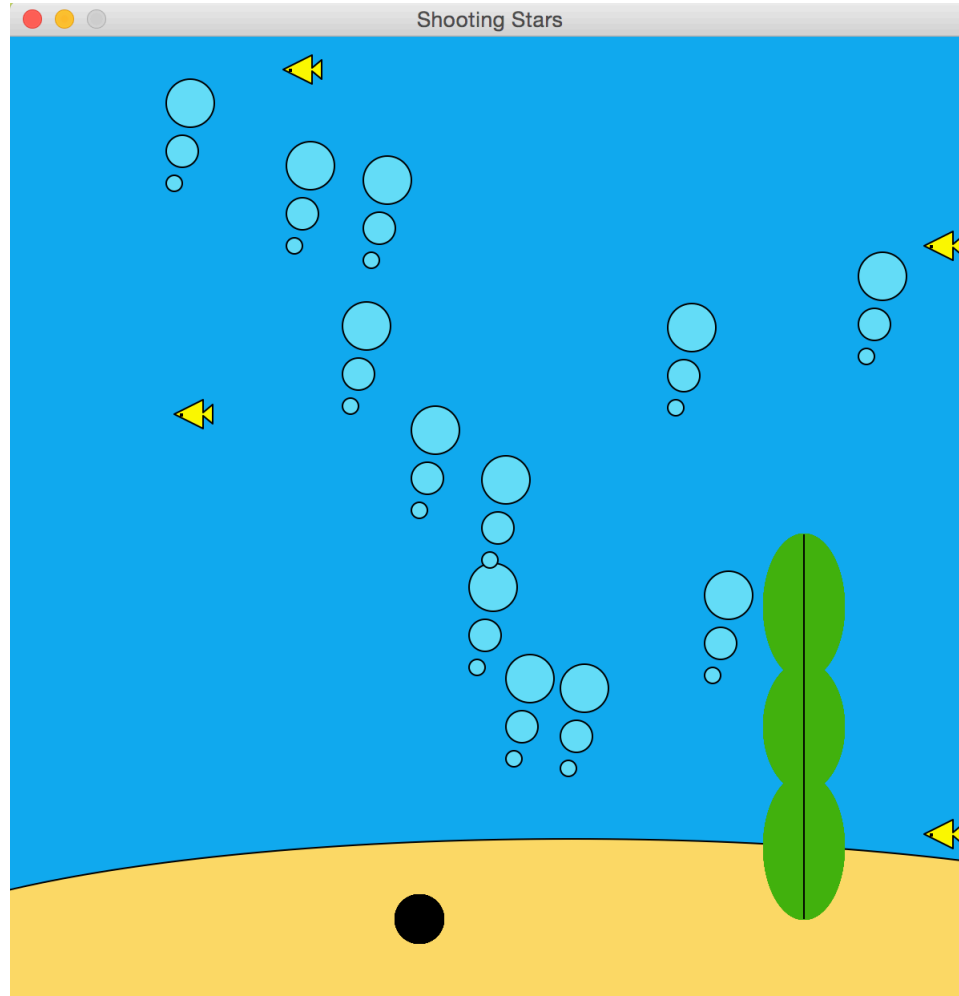
Homework 7 – Rebecca



Homework 7 – Cai



Homework 7 – Maddie



Finish 99 number activity

Step 1: make the dictionary

- + Find your random partner and introduce yourselves
- + In main, write some code that will ask the user for their 99 number and their name (two questions)
- + Use the first two and last two digits for speed and privacy
- + Add the 99 number (key) and name (value) to a dictionary that will keep track of individuals using their 99 numbers

```
>>>  
Enter your 99 number: 9995  
Enter your name: Sara Mathieson  
{9995: 'Sara Mathieson'}  
>>>
```

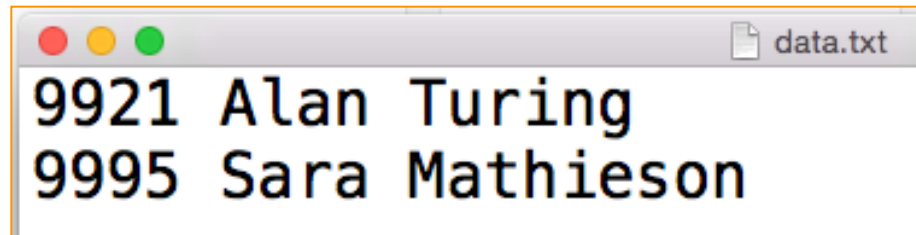
Step 2: use a while loop

- + Create a while loop that will keep asking for more 99 numbers and student names (use you and your partner's info)
- + Create a way to stop the while loop (i.e. user enters -1 for their number or "stop" for their name)

```
>>>
Enter your 99 number: 9995
Enter your name: Sara Mathieson
Enter your 99 number: 9921
Enter your name: Alan Turing
Enter your 99 number: -1
{9921: 'Alan Turing', 9995: 'Sara Mathieson'}
>>>
```

Step 3: write dictionary data to a file

- + After the while loop is over, write each number and name to a file using a loop over the keys of the dictionary



```
9921 Alan Turing
9995 Sara Mathieson
```

The image shows a screenshot of a text editor window titled "data.txt". The window contains two lines of text: "9921 Alan Turing" and "9995 Sara Mathieson". The text is displayed in a monospaced font.

Informal quiz: discuss with a partner

- 1) What built-in types have we studied this semester?
- 2) Which of those types are mutable?
- 3) If I call `binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`, how many times will `binary_search(..)` be called?
- 4) See the constructor below. How many instance variables? What is going on?

```
class Course:
    def __init__(self, title):
        tokens1 = title.split()
        self.dept = tokens1[0]
        self.number = int(tokens1[1][:-1])

        tokens2 = title.split(": ")
        self.name = tokens2[1]

        self.student_lst = []
        self.time = None
```

```
cs = Course("CSC 111: Introduction to Computer Science")
```

Informal quiz: discuss with a partner

- 1) What built-in types have we studied this semester?

int, str, list, bool, float, dict, tuple, "file"

Informal quiz: discuss with a partner

- 1) What built-in types have we studied this semester?

int, str, list, bool, float, dict, tuple, "file"

- 2) Which of those types are mutable?

list, dict, "file" (when using write)

Informal quiz: discuss with a partner

3) If I call `binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`,
how many times will `binary_search(..)` be called?

```
binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])
```


Informal quiz: discuss with a partner

3) If I call `binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`,
how many times will `binary_search(..)` be called?

`binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`

`binary_search("e", ["a", "c", "g"])`

Informal quiz: discuss with a partner

3) If I call `binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`,
how many times will `binary_search(..)` be called?

`binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`

`binary_search("e", ["a", "c", "g"])`

`binary_search("e", ["c", "g"])`

Informal quiz: discuss with a partner

3) If I call `binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`,
how many times will `binary_search(..)` be called?

`binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`

`binary_search("e", ["a", "c", "g"])`

`binary_search("e", ["c", "g"])`

`binary_search("e", ["c"])`

Informal quiz: discuss with a partner

3) If I call `binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`,
how many times will `binary_search(..)` be called?

`binary_search("e", ["a", "c", "g", "i", "s", "t", "w"])`

`binary_search("e", ["a", "c", "g"])`

`binary_search("e", ["c", "g"])`

`binary_search("e", ["c"])`

4 times

Course class

4) See the constructor below. How many instance variables? What is going on?

```
class Course:

    def __init__(self, title):
        tokens1 = title.split()
        self.dept = tokens1[0]
        self.number = int(tokens1[1][:-1])

        tokens2 = title.split(": ")
        self.name = tokens2[1]

        self.student_lst = []
        self.time = None
```

```
cs = Course("CSC 111: Introduction to Computer Science")
```

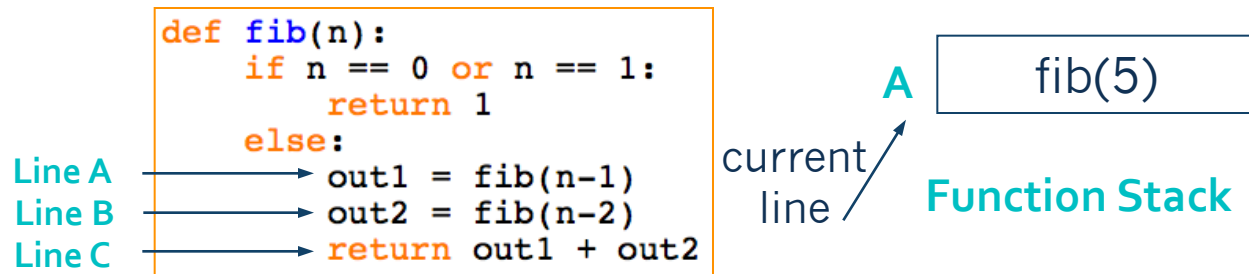
Recursion

Recursion

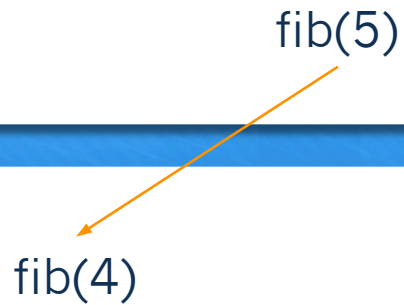
- + A recursive function must call itself
- + Usually it will call itself on a simpler/smaller/different version of the problem
- + Must have a way of stopping the recursion (base case)
- + If we return in the base case, we must return in the recursive call
- + Going “down” to the base case, passing the answer back “up” through the recursive calls

Fibonacci Function Stack

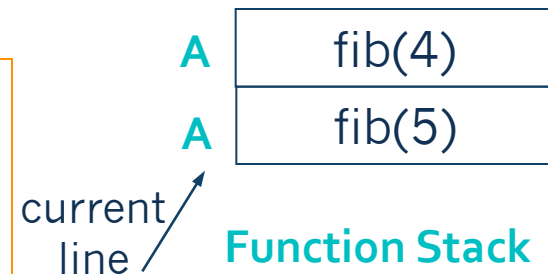
fib(5)



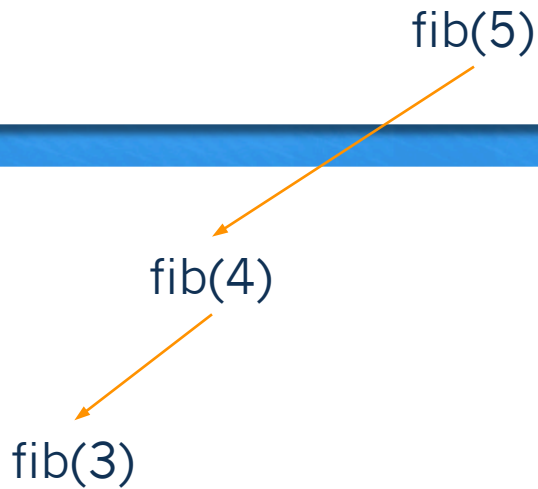
Fibonacci Function Stack



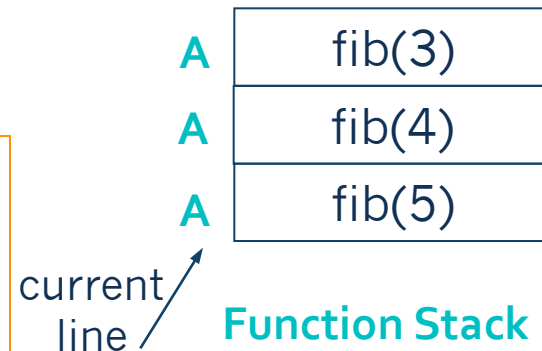
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



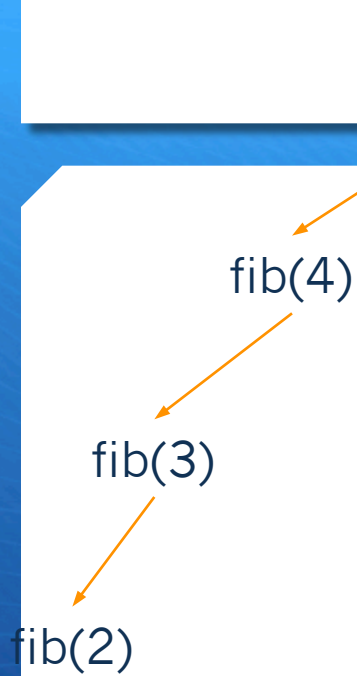
Fibonacci Function Stack



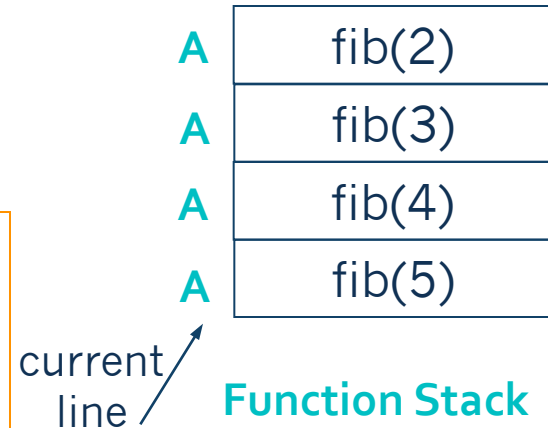
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



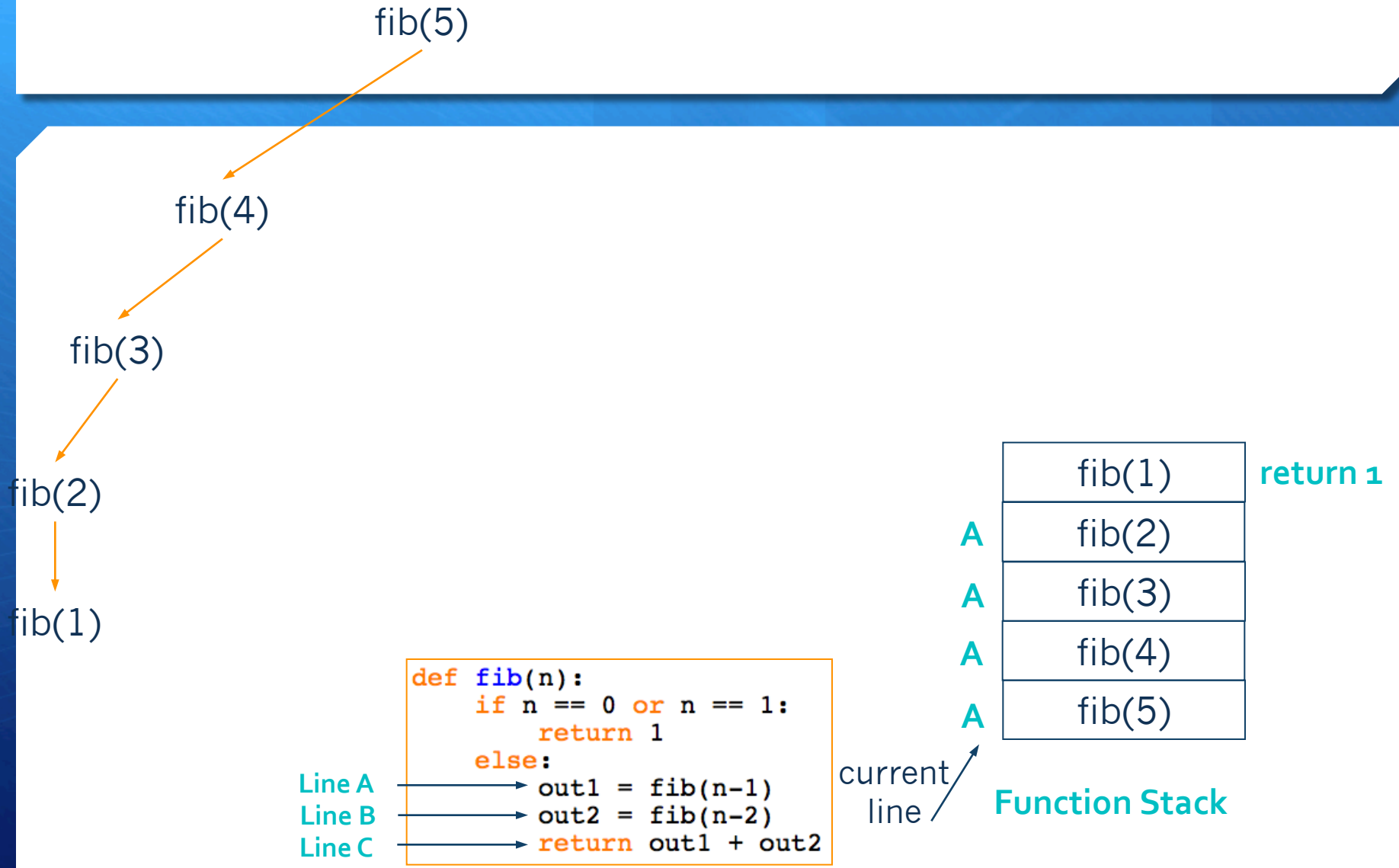
Fibonacci Function Stack



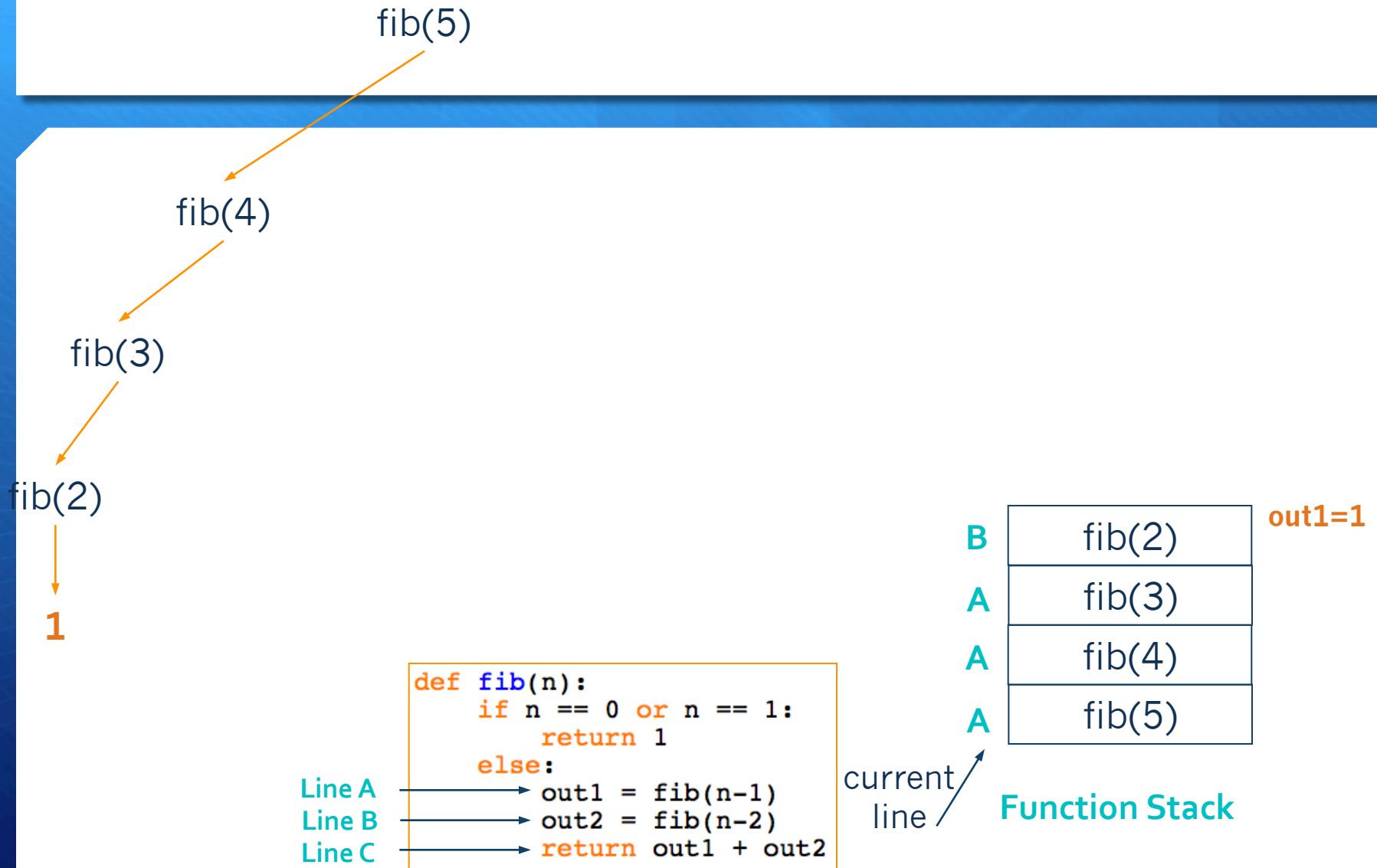
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



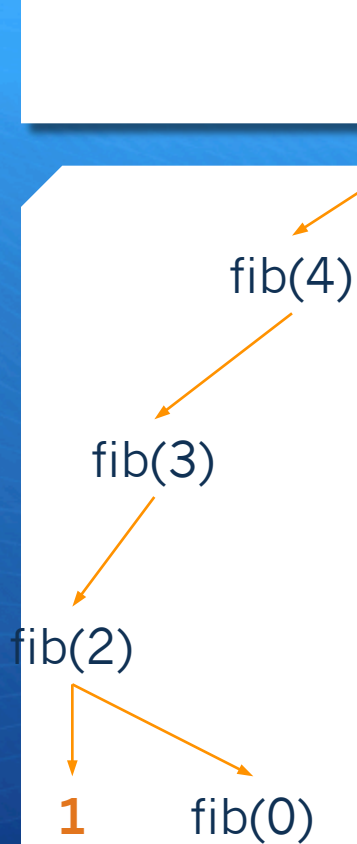
Fibonacci Function Stack



Fibonacci Function Stack



Fibonacci Function Stack



fib(5)

fib(4)

fib(3)

fib(2)

1

fib(0)

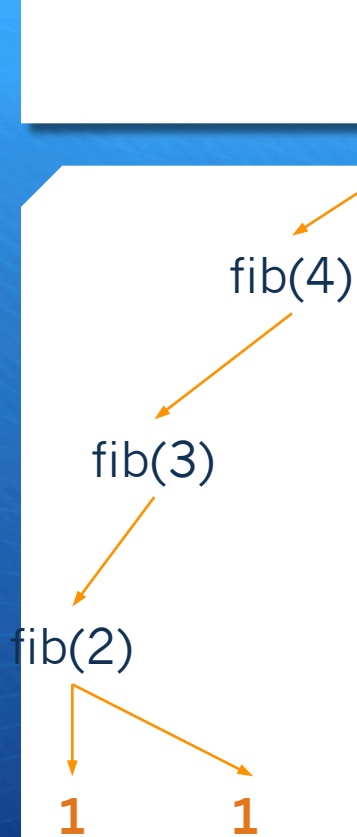
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```

	fib(0)	return 1
B	fib(2)	out1=1
A	fib(3)	
A	fib(4)	
A	fib(5)	

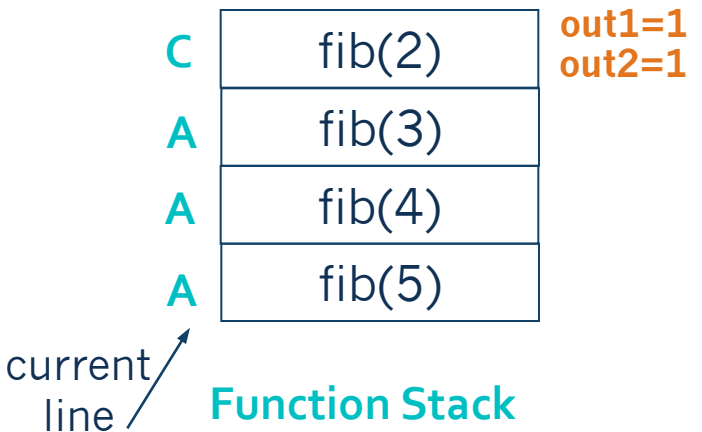
current line ↗

Function Stack

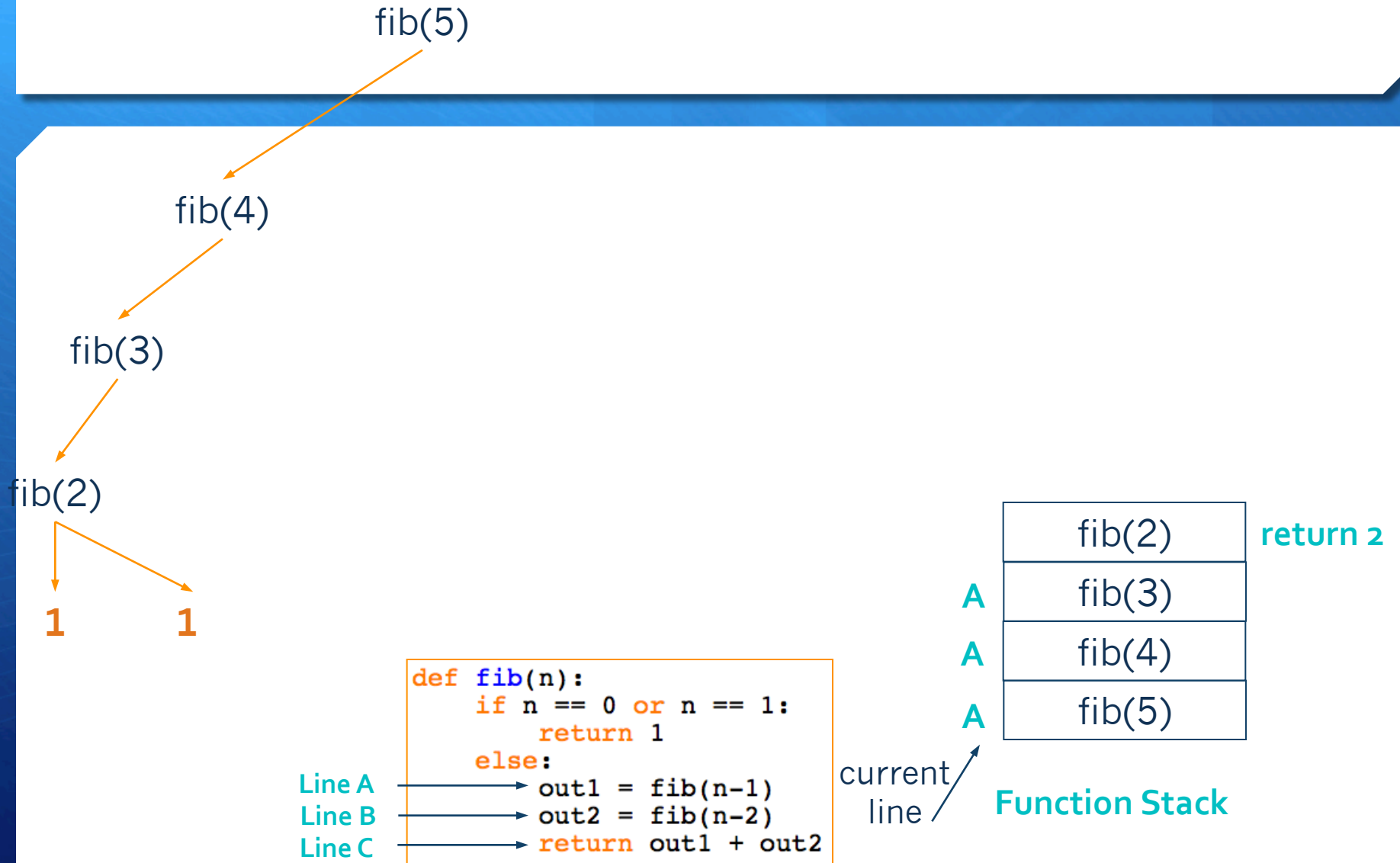
Fibonacci Function Stack



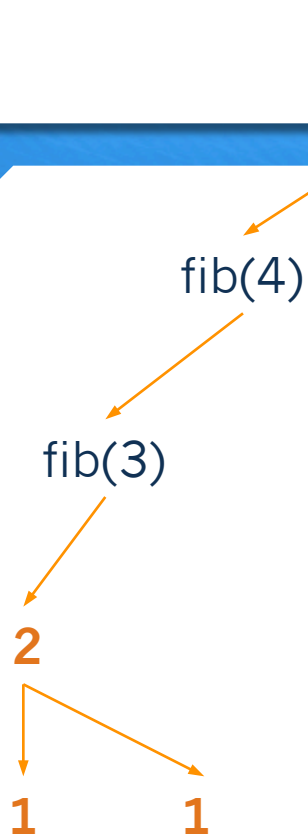
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



Fibonacci Function Stack



Fibonacci Function Stack



fib(5)

fib(4)

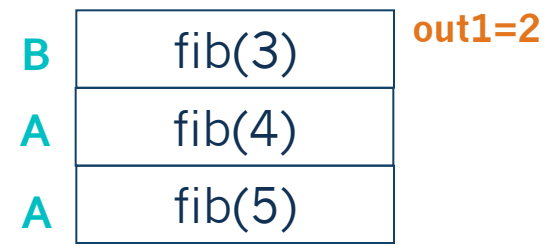
fib(3)

2

1

1

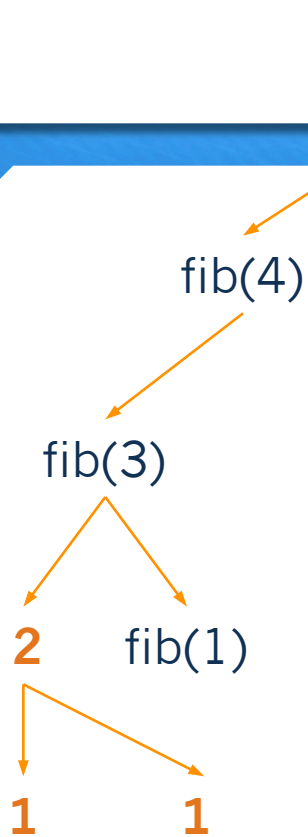
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



current line ↗

Function Stack

Fibonacci Function Stack



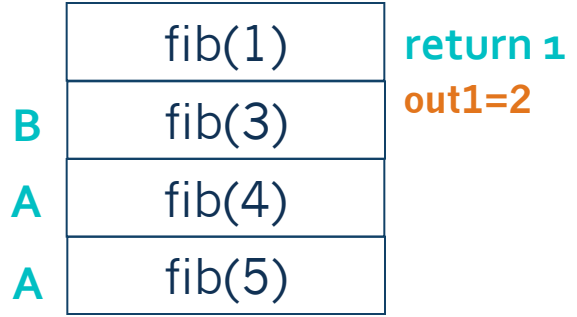
fib(5)

fib(4)

fib(3)

2 fib(1)
1 1

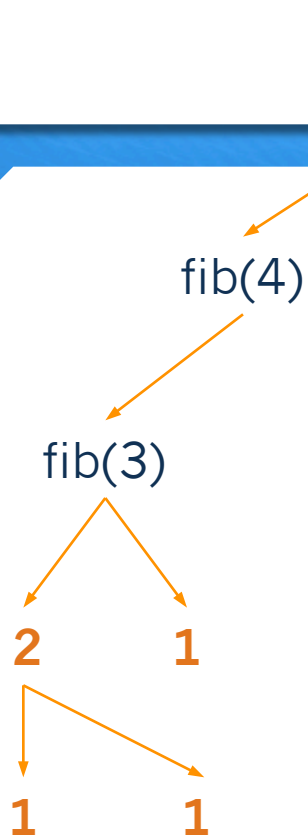
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



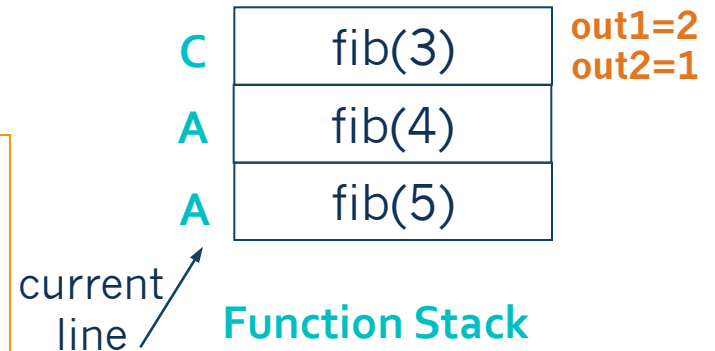
current line ↗

Function Stack

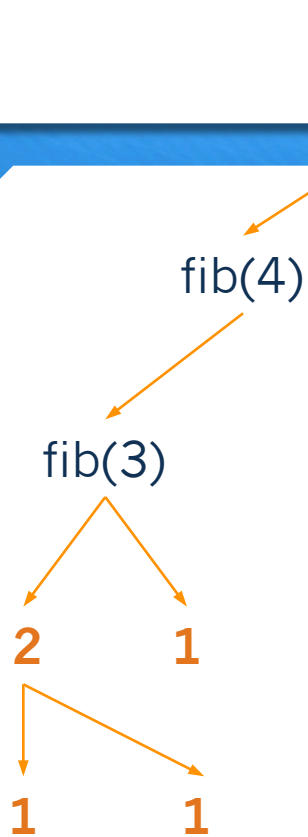
Fibonacci Function Stack



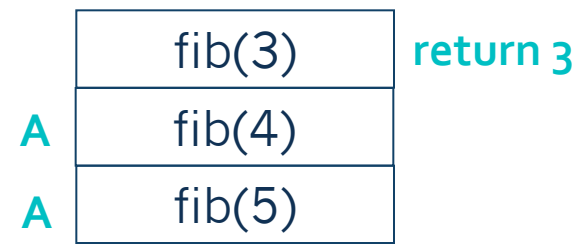
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



Fibonacci Function Stack



```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



current line ↗

Function Stack

Fibonacci Function Stack

fib(5)

fib(4)

3

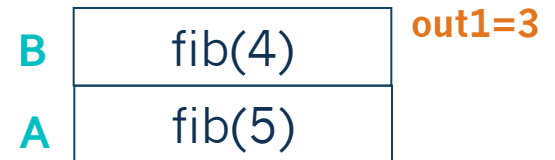
2

1

1

1

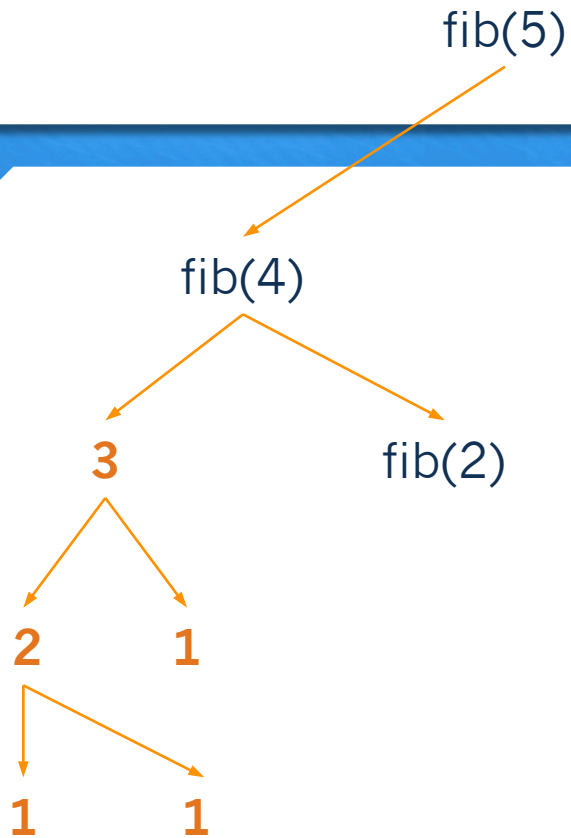
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



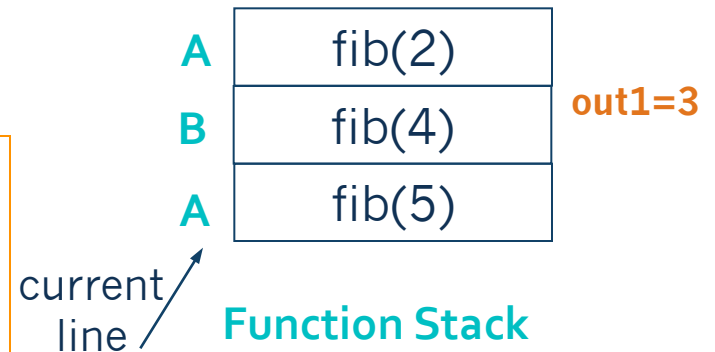
current line

Function Stack

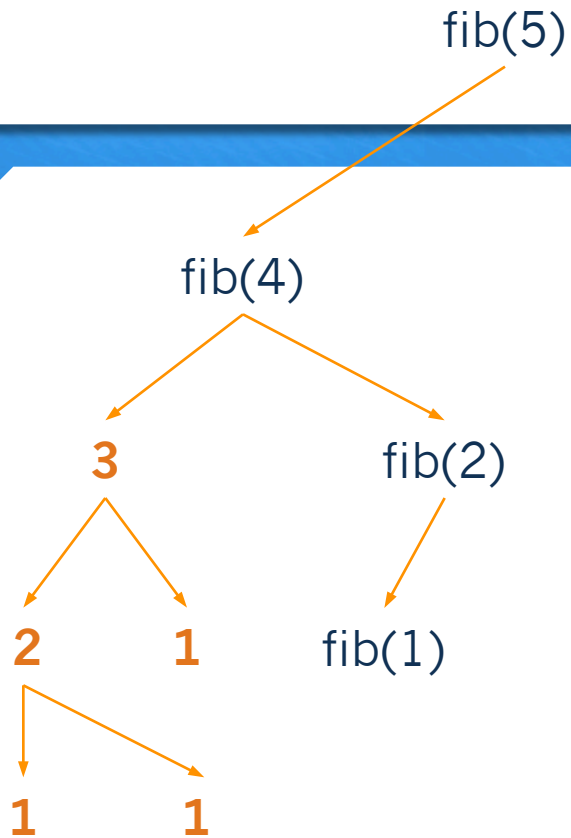
Fibonacci Function Stack



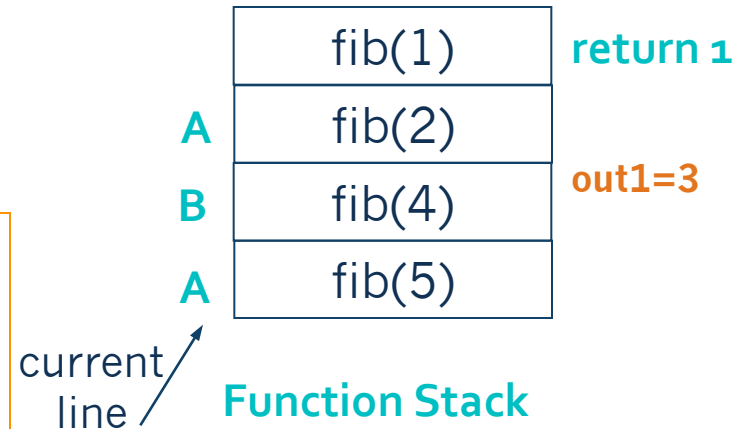
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



Fibonacci Function Stack



```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



Fibonacci Function Stack

fib(5)

fib(4)

3

fib(2)

2

1

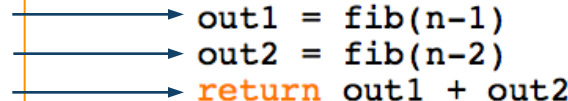
1

1

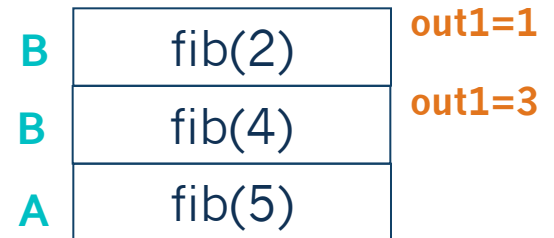
1

```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        out1 = fib(n-1)  
        out2 = fib(n-2)  
        return out1 + out2
```

Line A
Line B
Line C

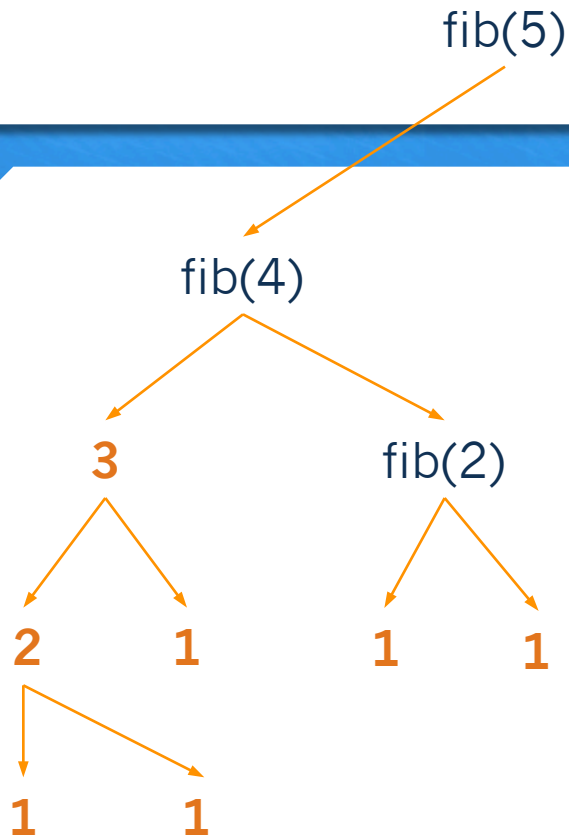


current line

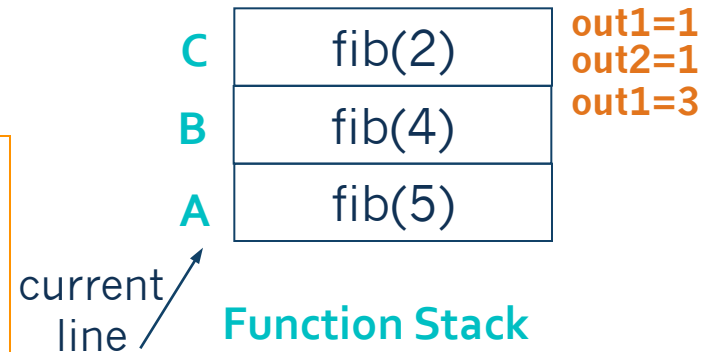


Function Stack

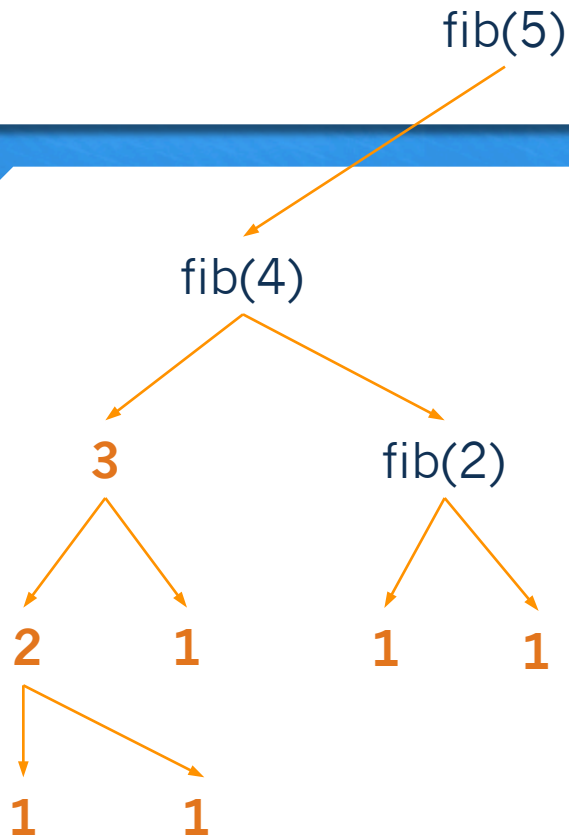
Fibonacci Function Stack



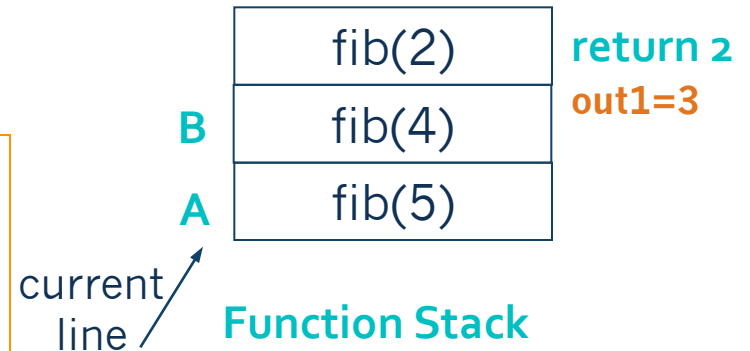
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



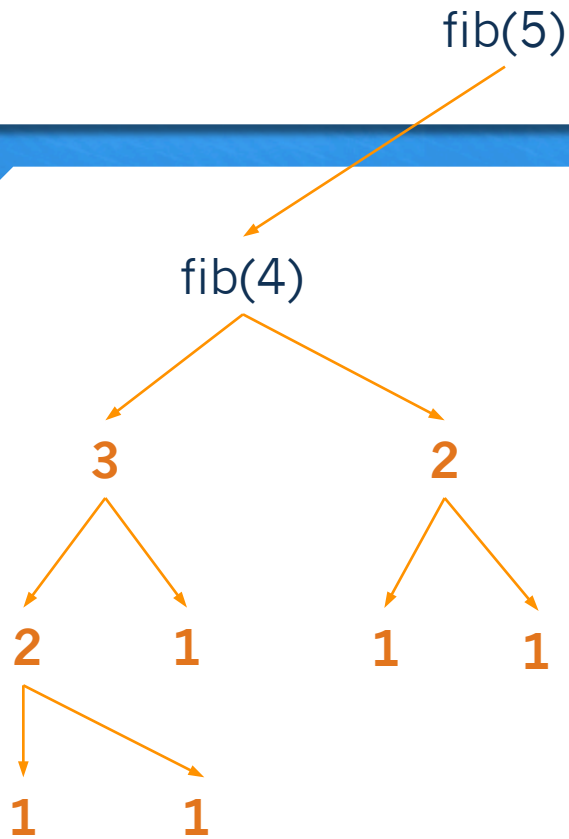
Fibonacci Function Stack



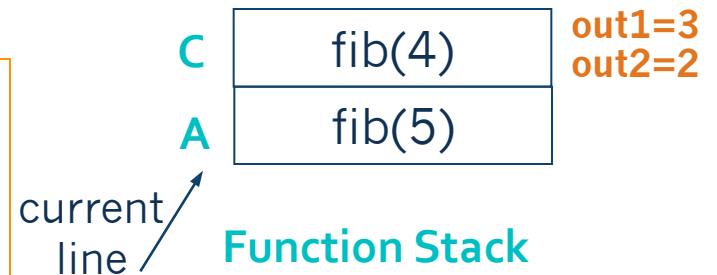
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



Fibonacci Function Stack



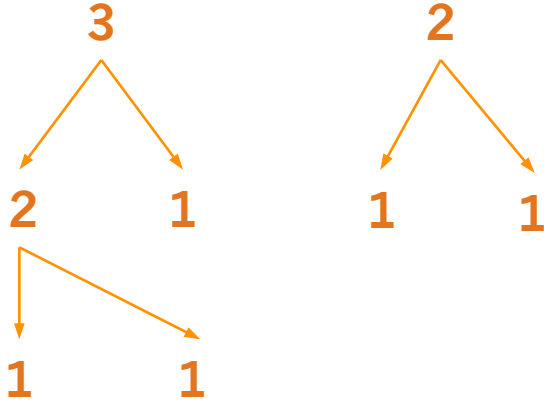
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



Fibonacci Function Stack

fib(5)

fib(4)

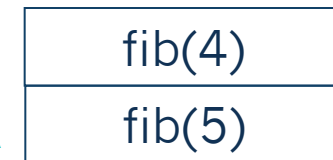


```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        out1 = fib(n-1)  
        out2 = fib(n-2)  
        return out1 + out2
```

Line A
Line B
Line C



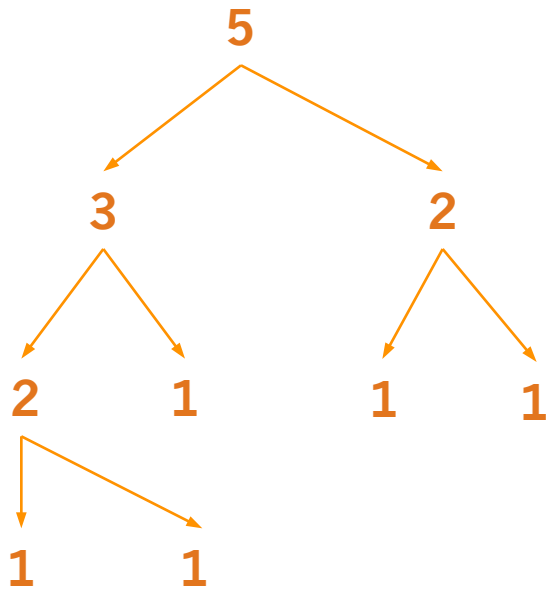
current line / A



Function Stack

Fibonacci Function Stack

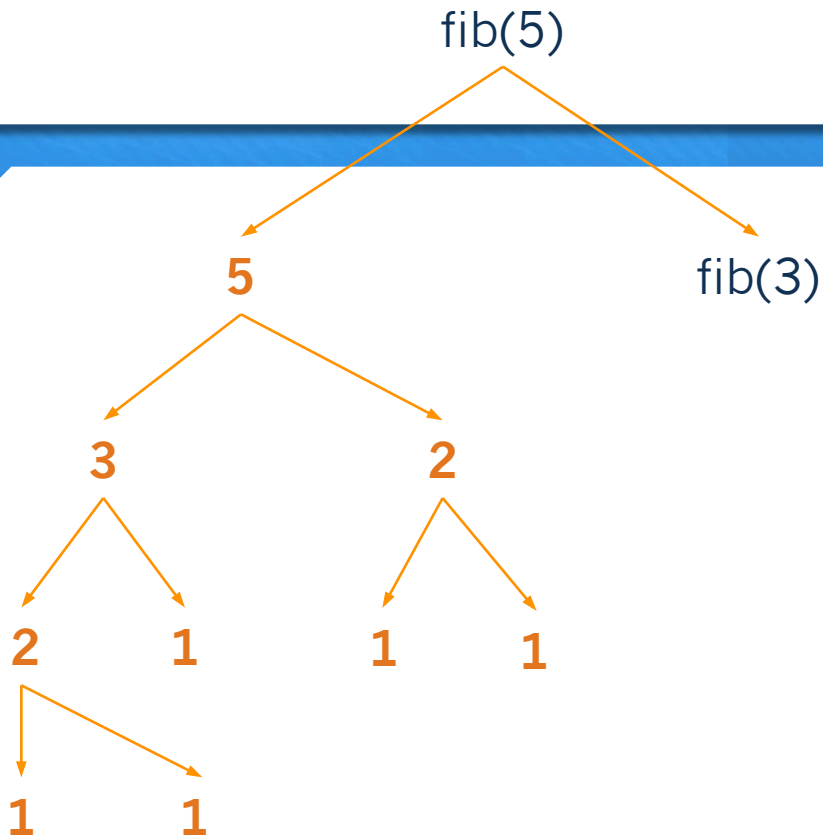
fib(5)



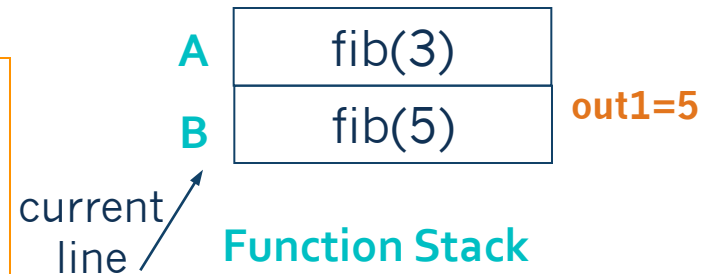
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



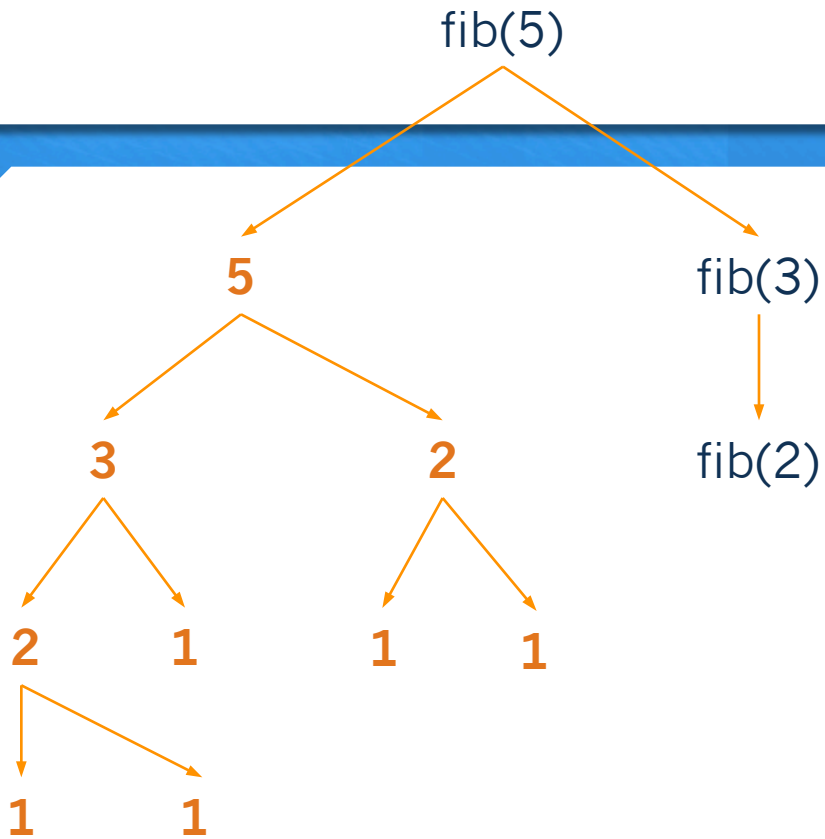
Fibonacci Function Stack



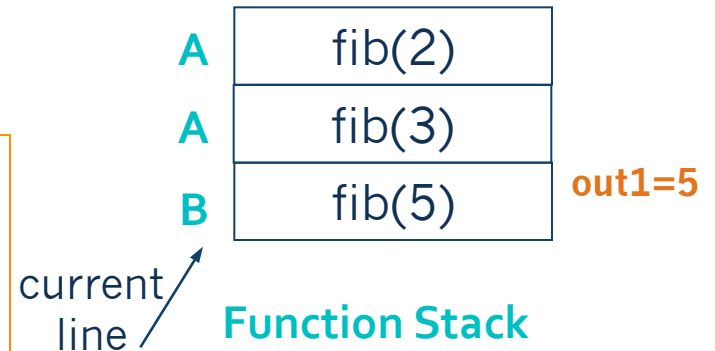
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



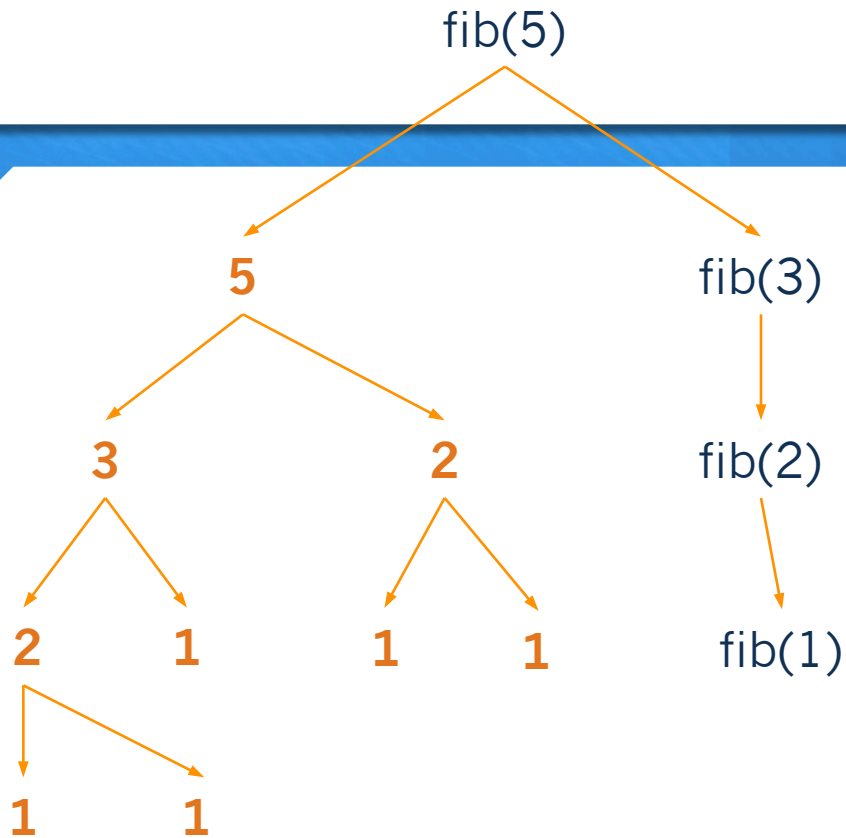
Fibonacci Function Stack



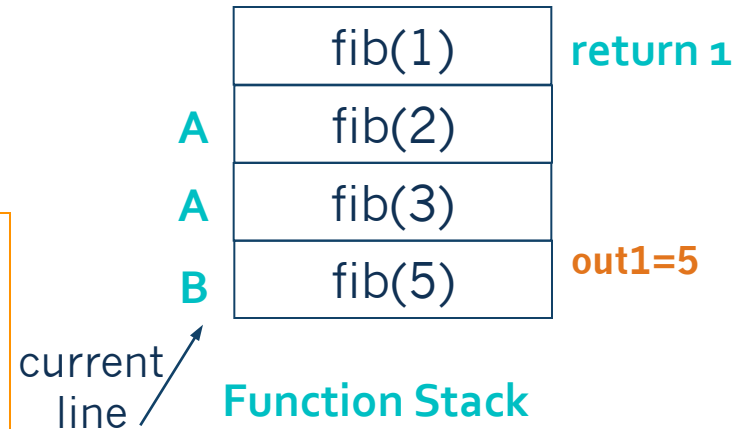
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



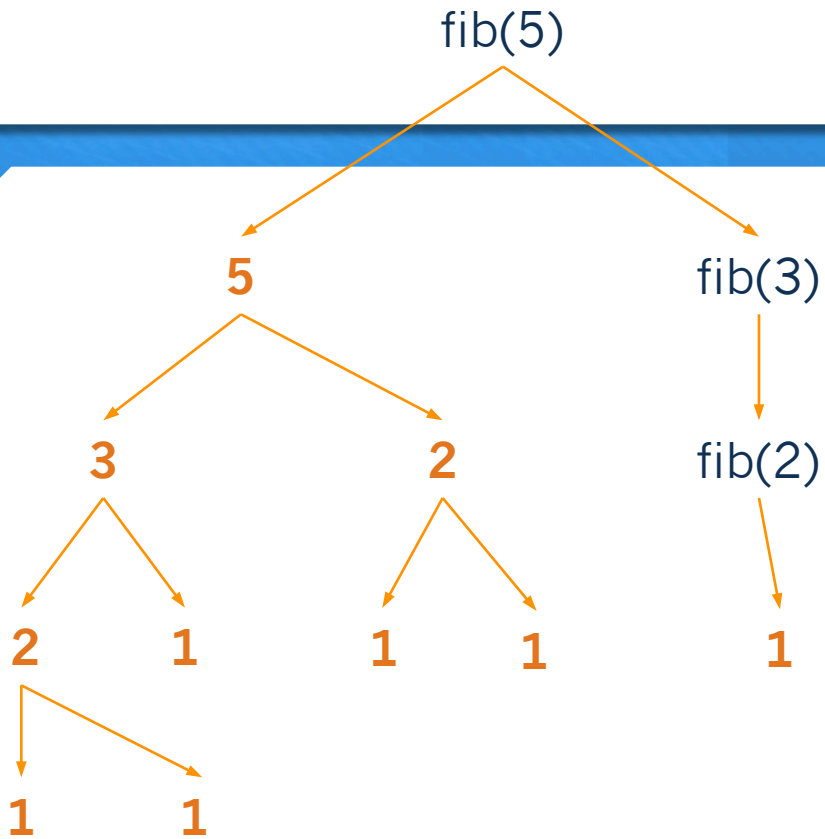
Fibonacci Function Stack



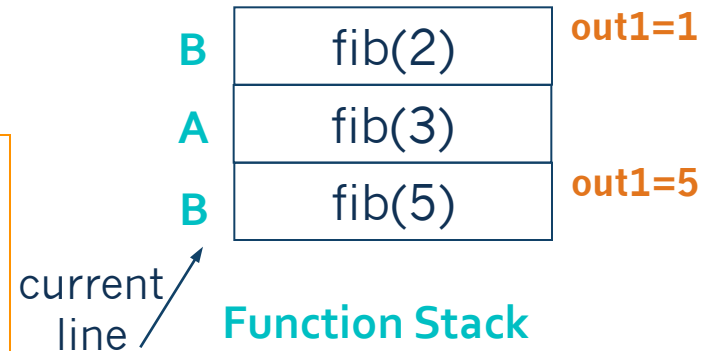
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



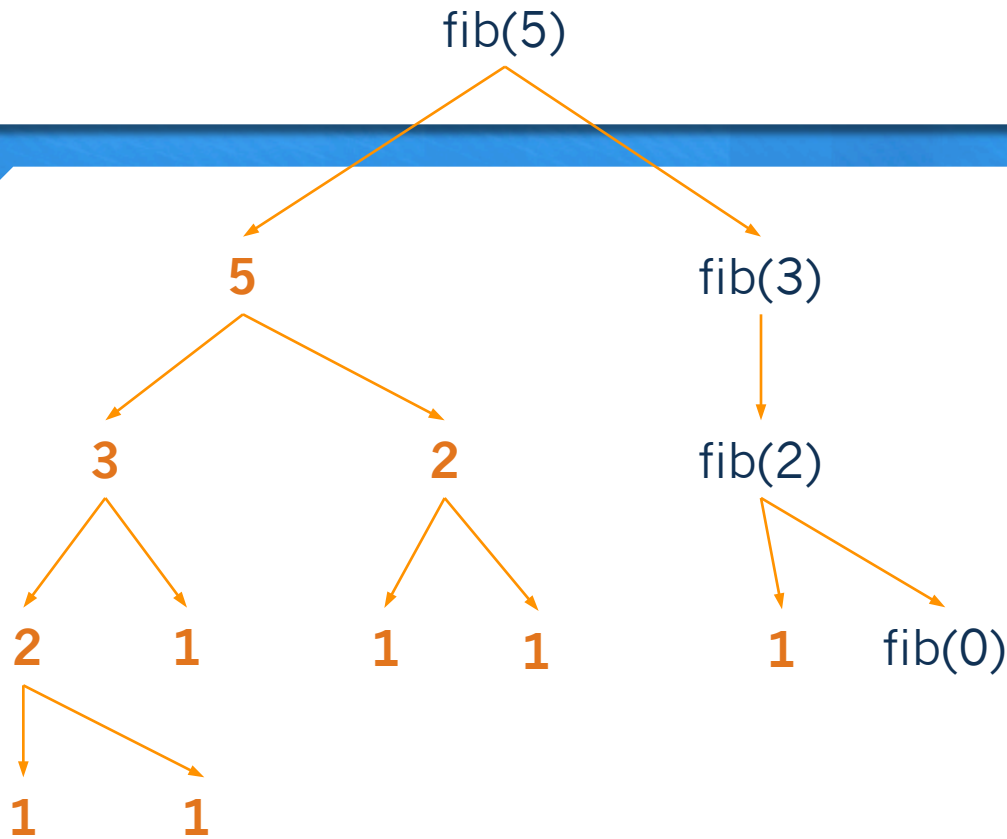
Fibonacci Function Stack



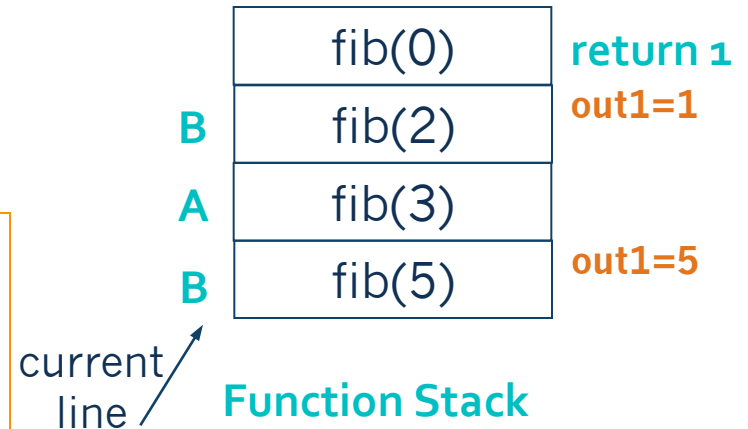
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



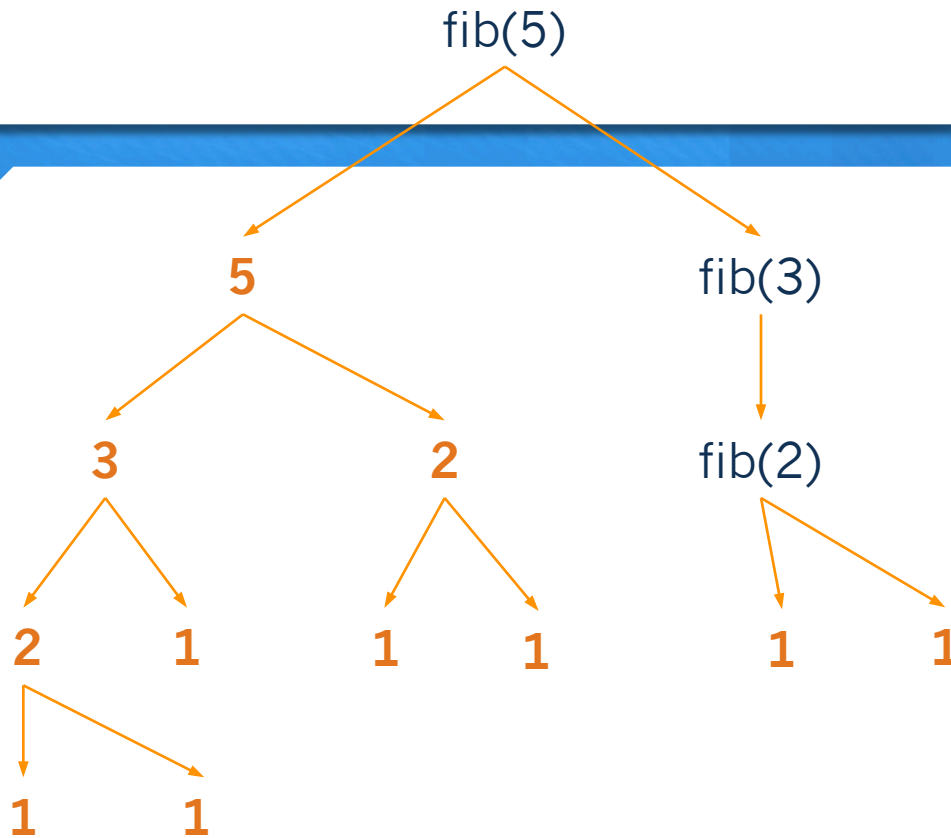
Fibonacci Function Stack



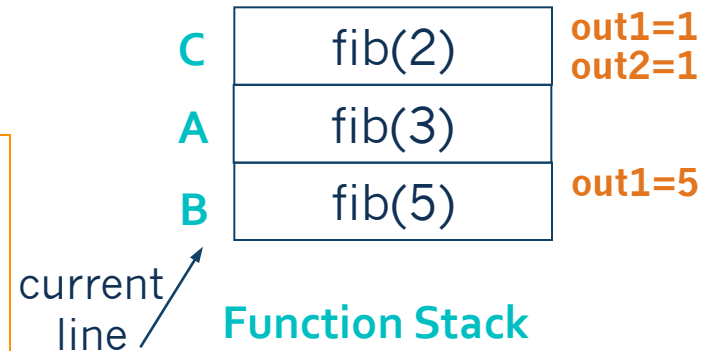
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



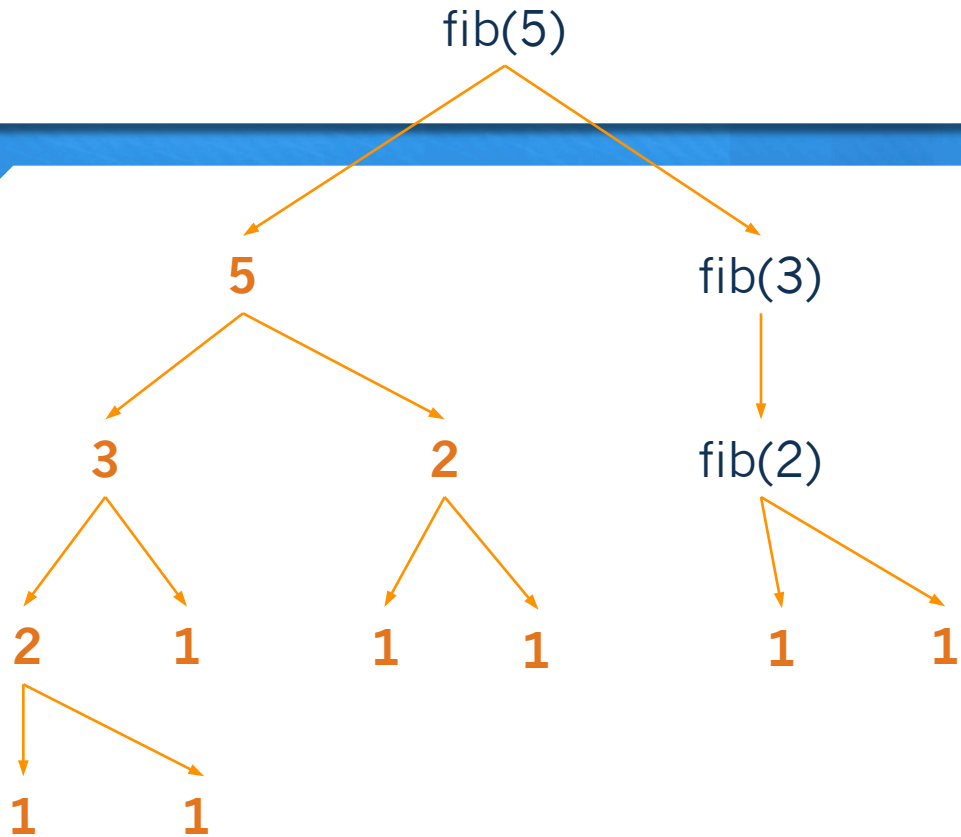
Fibonacci Function Stack



```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



Fibonacci Function Stack

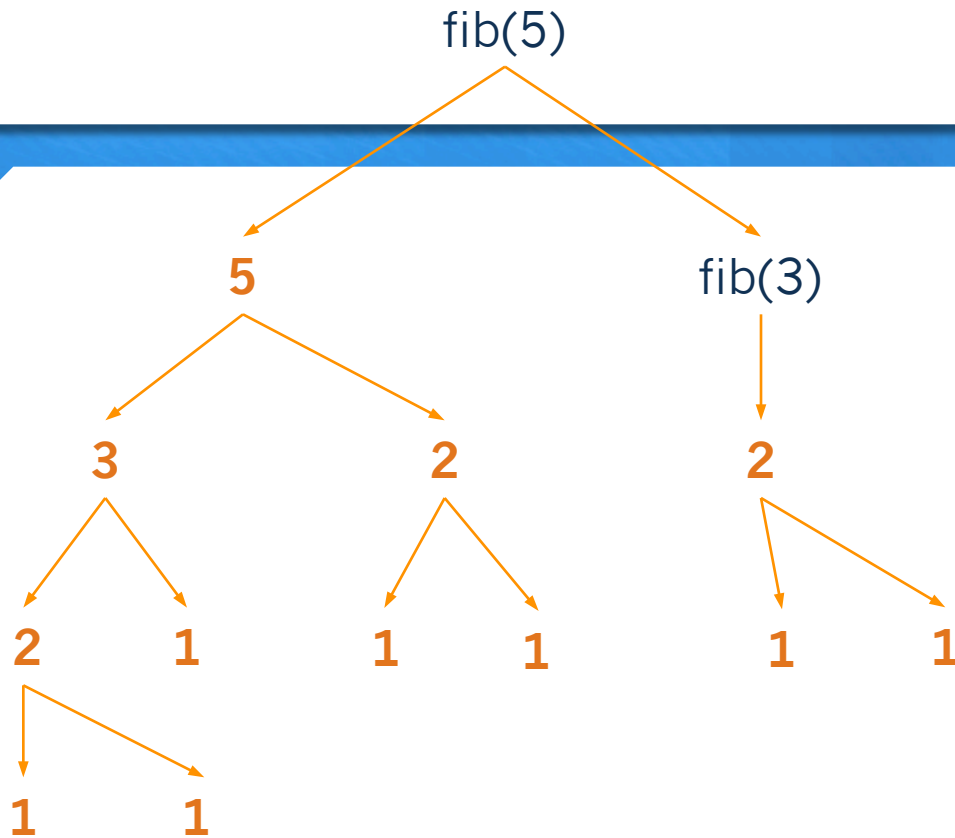


```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```

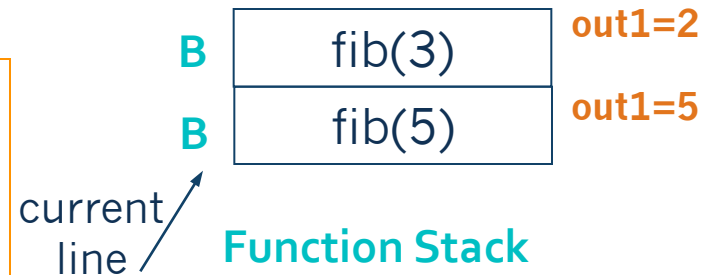


Function Stack

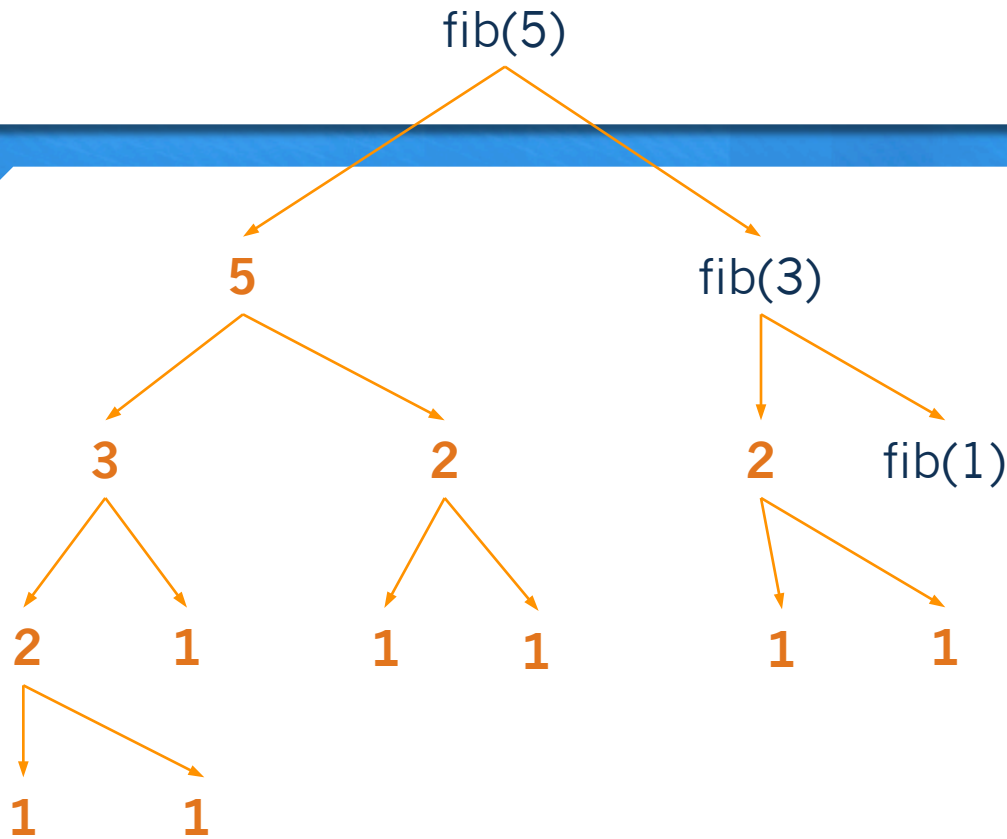
Fibonacci Function Stack



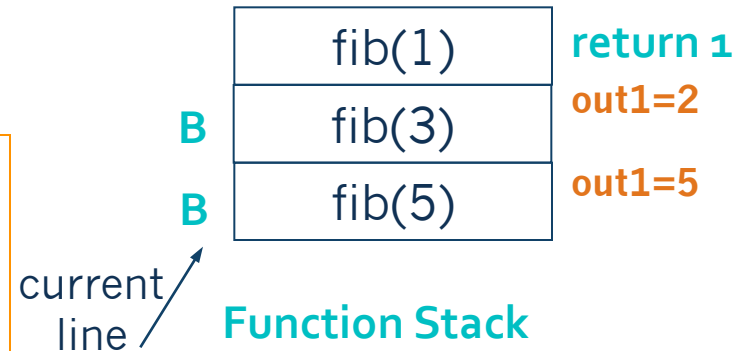
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



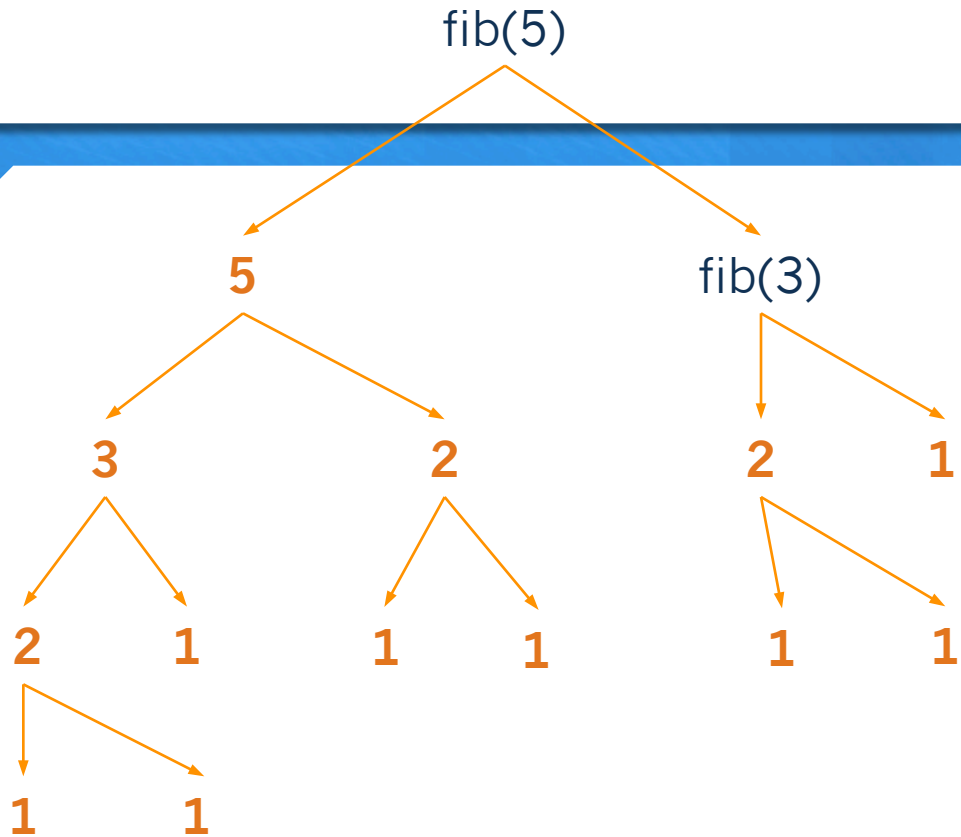
Fibonacci Function Stack



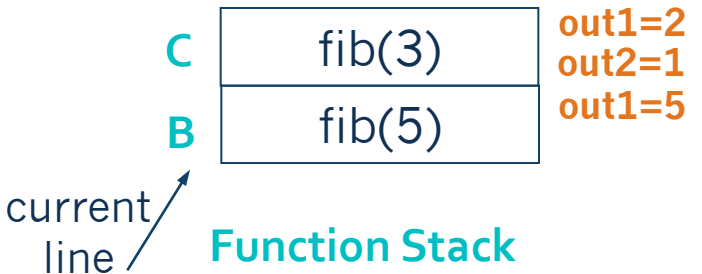
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



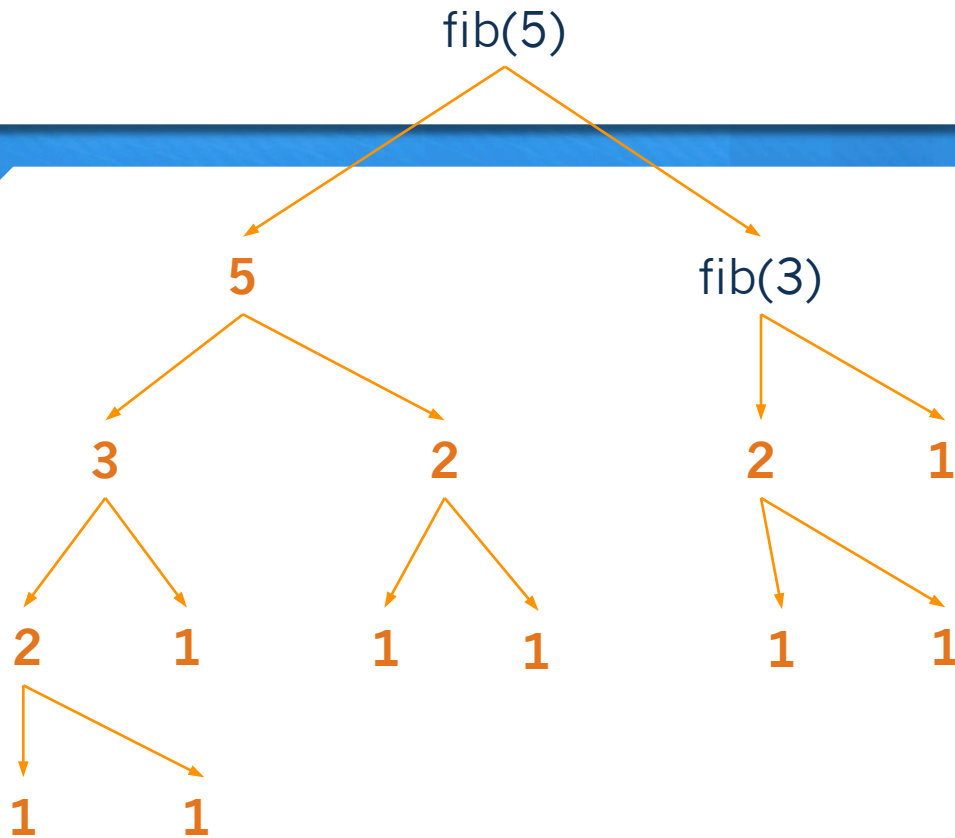
Fibonacci Function Stack



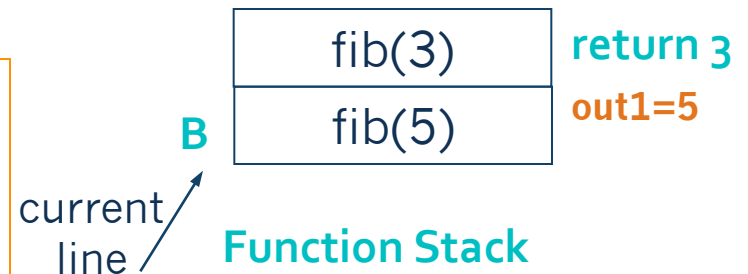
```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



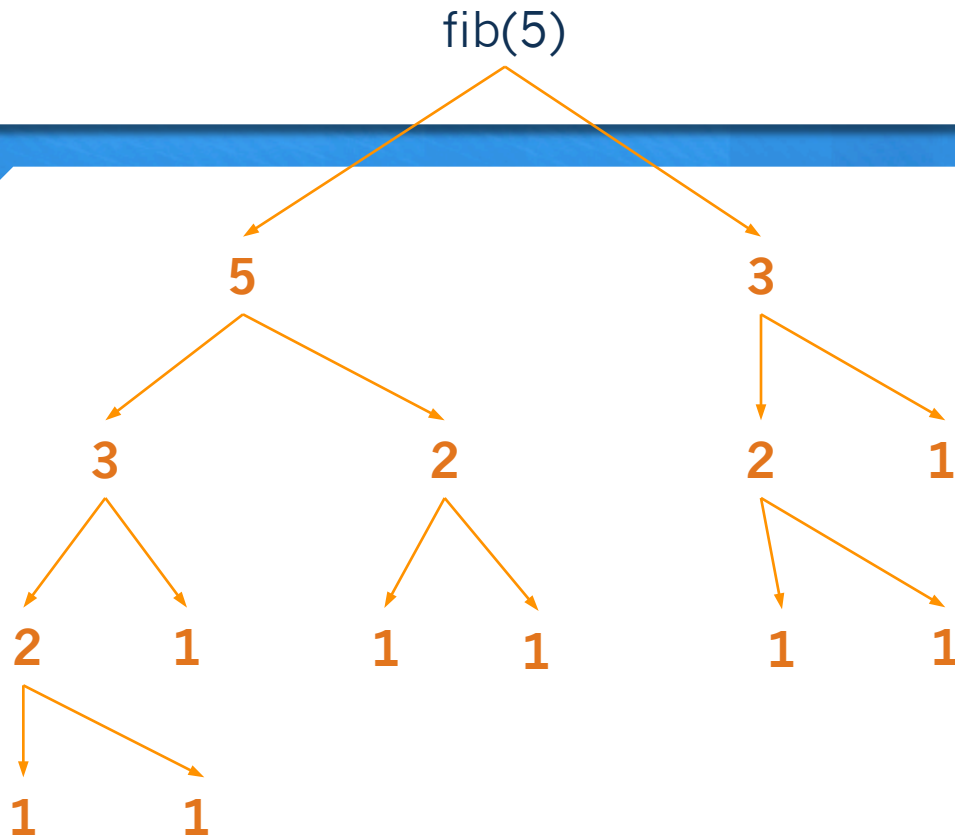
Fibonacci Function Stack



```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



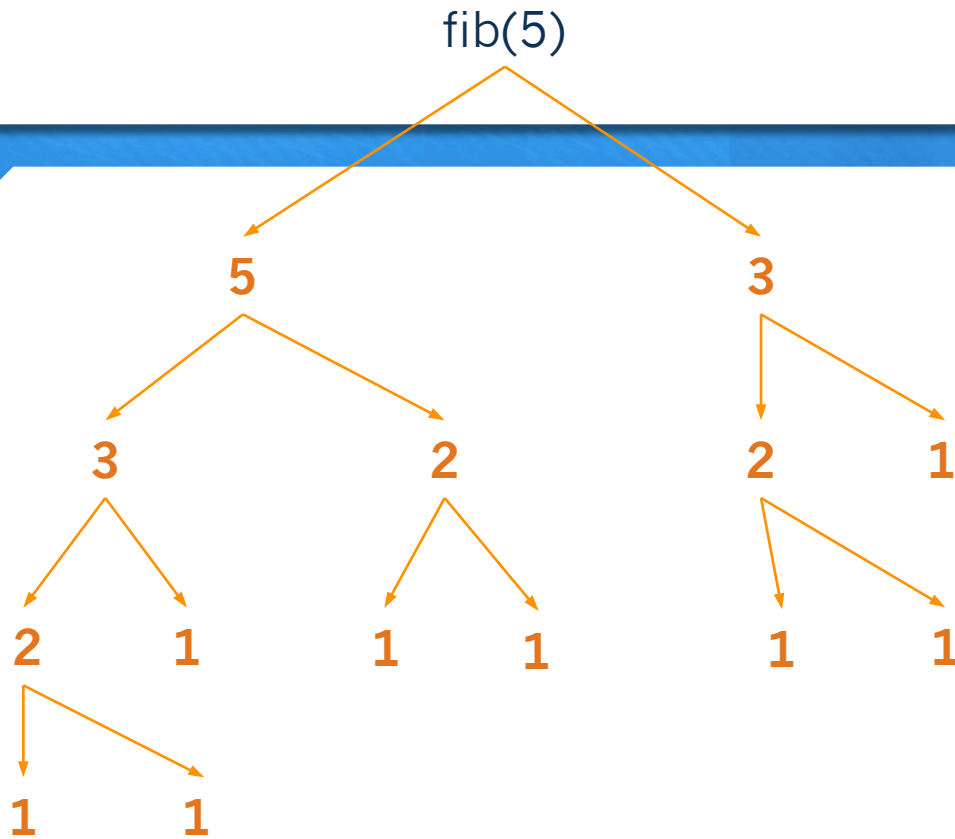
Fibonacci Function Stack



```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```



Fibonacci Function Stack

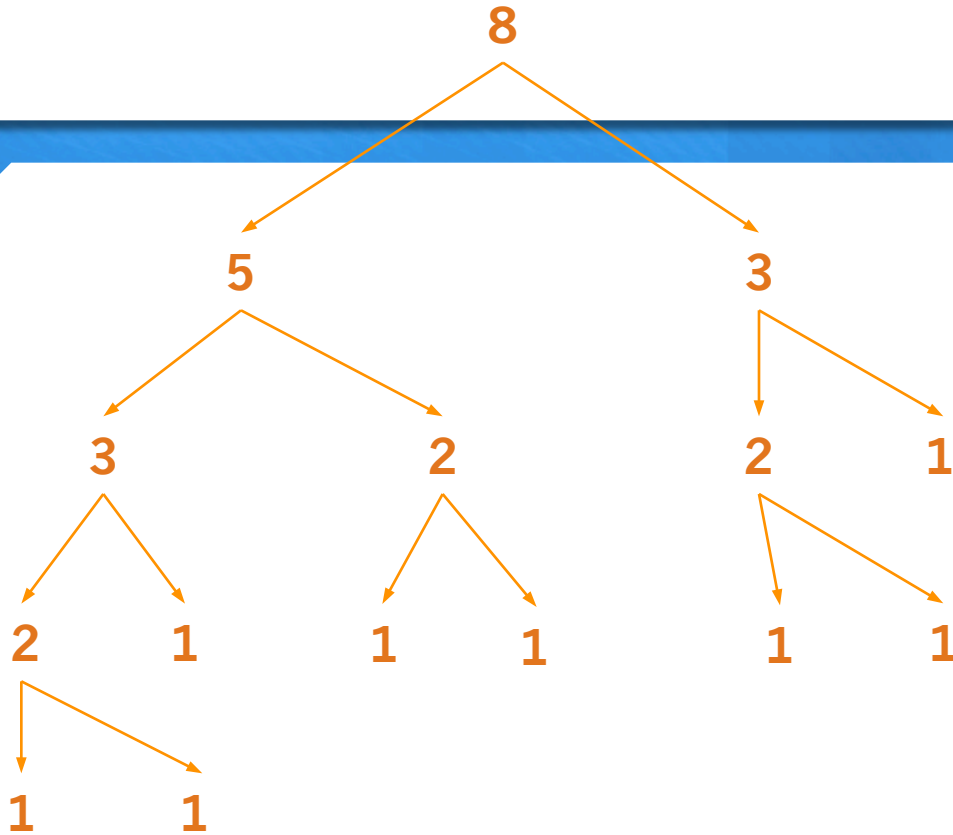


```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```

current
line ↗



Fibonacci Function Stack

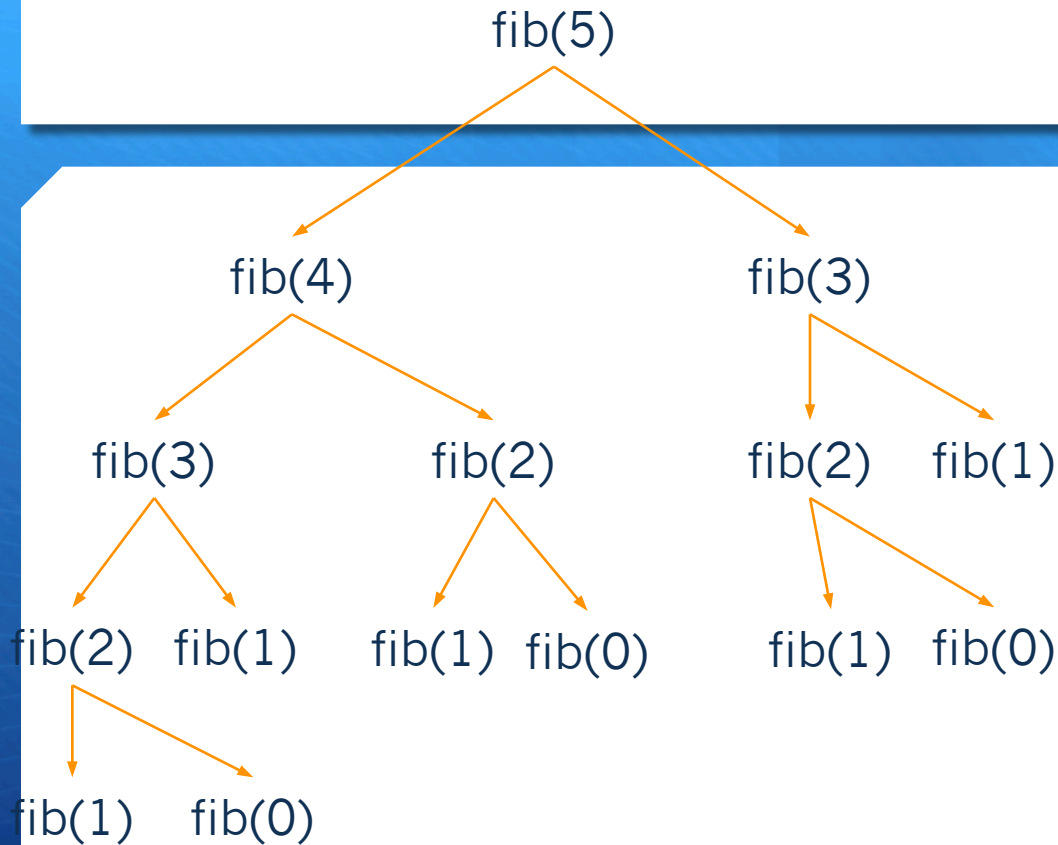


```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        Line A → out1 = fib(n-1)  
        Line B → out2 = fib(n-2)  
        Line C → return out1 + out2
```

empty!

Function Stack

Fibonacci Tree with Function Calls



```
def fib(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        out1 = fib(n-1)  
        out2 = fib(n-2)  
        return out1 + out2
```

Line A →
Line B →
Line C →

Lab 11 (sort and search)

```
def binary_search(item, lst):
    #print(lst)

    # base case
    if len(lst) == 1:
        return lst[0]

    index = int(len(lst)/2)
    middle = lst[index]

    # recursion
    if item < middle:
        return binary_search(item, lst[:index])
    else:
        return binary_search(item, lst[index:])
```


Classes

Classes

- + Goal of classes: encapsulate data (**instance variables**) and operations on data (**methods**) in one structure we can represent with single variable names

Classes

- + Goal of classes: encapsulate data (**instance variables**) and operations on data (**methods**) in one structure we can represent with single variable names
- + **list, str, dict** are also [special] classes, and instances of these classes have methods associated with them

Classes

- + Goal of classes: encapsulate data (**instance variables**) and operations on data (**methods**) in one structure we can represent with single variable names
- + **list**, **str**, **dict** are also [special] classes, and instances of these classes have methods associated with them
- + When we say **x = [3,5,6]** and then say **print(x)**, calling the **list** **__str__** method internally

Classes

- + Goal of classes: encapsulate data (**instance variables**) and operations on data (**methods**) in one structure we can represent with single variable names
- + **list**, **str**, **dict** are also [special] classes, and instances of these classes have methods associated with them
- + When we say **x = [3,5,6]** and then say **print(x)**, calling the **list __str__** method internally
- + **Fish**, **Car**, etc are no different or less special than **Circle**, **Point**

Template for classes

```
class MyClass:
```

```
    def __init__(self,         ,         ):
```

```
        self.        
```

```
        self.        
```

```
    def method1(self,         ):
```

```
        self.
```

```
    def method2(self,         ,         ):
```

```
        self.
```



```
def function1(        ,         ,         ):
```

```
def function2(        ,         ):
```


```
def main():
```

Template for classes



```
class MyClass:
```

```
    def __init__(self, , ):
```

```
        self.   
        self.
```

```
    def method1(self, ):
```

```
        self.
```

```
    def method2(self, , ):
```

```
        self.
```

```
def function1(, , ):
```

```
    
```

```
def function2(, ):
```



```
    
```

```
def main():
```


```
    
```

Template for classes



```
class MyClass:
```

```
    def __init__(self, , ):
```

```
        self.   
        self.
```

```
    def method1(self, ):
```

```
        self.
```

```
    def method2(self, , ):
```

```
        self.
```

```
def function1(, , ):
```

```
       
     
```

```
def function2(, ):
```



```
       
    
```

```
def main():
```


```
    
```


Template for classes



```
class MyClass:
```

```
    def __init__(self, , ):
```

```
        self.    
        self.   
```

```
    def method1(self, ):
```

```
        self.   
        
```

```
    def method2(self, , ):
```

```
        self.   
         
```

```
    def function1(, , ):
```

```
           
         
```

```
    def function2(, ):
```



```
           
        
```






```
    def main():
```


```
        
```

Template for classes



```
class MyClass:
```

```
    def __init__(self, , ):
```

```
        self.   
        self.   
            
```

```
    def method1(self, ):
```

```
        self.   
             
```

```
    def method2(self, , ):
```

```
        self.   
              
```

```
    def function1(, , ):
```

```
           
             
```

```
    def function2(, ):
```

```
           
            
```

```
    def main():
```

```
        
```

Projectile class

Constructor:
Special
method

update(dt) is
kind of like
move(..) for
graphics
objects

Getters

```
# use CamelCase for class names (no underscores)
class Projectile:

    def __init__(self, height, velocity, angle):

        # two instance variables to keep track of position
        self.x = 0
        self.y = height

        # two instance variables to keep track of velocity
        theta = angle*math.pi/180 # OR math.radians(angle)
        self.xvel = velocity * math.cos(theta)
        self.yvel = velocity * math.sin(theta)

    def update(self, dt):
        """Update the position and velocity based on:
        dt, a small amount of time in seconds."""

        # update the (x,y) position
        self.x += (self.xvel * dt)
        self.y += (self.yvel * dt)

        # update y-velocity (don't need to update x-velocity)
        self.yvel -= (9.8 * dt) # gravity: 9.8 m/(s^2)

    def getX(self):
        """Return the x position of the projectile."""
        return self.x

    def getY(self):
        """Return the y position of the projectile."""
        return self.y
```

Methods:
__init__(..)
update(..)
getX()
getY()

Instance
variables:
x
y
xvel
yvel

Projectile class

```
def main():  
  
    # either try out values or ask the user  
    height = eval(input("Enter height in meters: ")) # example: 1.6  
    velocity = eval(input("Enter velocity in m/s: ")) # example: 5  
    angle = eval(input("Enter angle in degrees: ")) # example: 40  
    dt = 0.01 # seconds  
  
    # create our projectile using the constructor  
    shotput = Projectile(height, velocity, angle)  
  
    # update the position while the projectile is above ground  
    while shotput.getY() >= 0:  
        shotput.update(dt)  
        print(shotput.getX(), shotput.getY()) # print current position
```

Lab 10 (with `__str__`)

```
class Matrix:

    def __init__(self, num_rows, num_cols):
        self.data = []
        for i in range(num_rows):
            self.data.append([0]*num_cols)

    def get_entry(self, i, j):
        return self.data[i][j]

    def set_entry(self, i, j, value):
        self.data[i][j] = value
```

List Comprehensions

List comprehensions

```
>>> # list comprehensions (ways of operating on a list without a for loop)
>>> lst1 = ["1","2","3","4"]
>>>
>>> # how to turn all elements to ints?
>>> lst2 = []
>>> for s in lst1:
>>>     lst2.append(int(s))

>>> lst2
[1, 2, 3, 4]
>>>
>>> # OR
>>>
>>> lst3 = [int(s) for s in lst1]
>>> lst3
[1, 2, 3, 4]
>>>
>>> lst4 = [x+10 for x in lst3]
>>> lst4
[11, 12, 13, 14]
>>>
>>> [fish.move() for fish in fish_lst]
```

Feedback

We are interested in thoughts about...

- Capping enrollments for 111
- What do you think about random partners during lab?
- The textbook: make optional in the future?
- Use of class time

Final Thoughts on CS