CS 106 INTRODUCTION TO DATA STRUCTURES

SPRING 2020

PROF. SARA MATHIESON
HAVERFORD COLLEGE

ADMIN

- **Lab 3 due Sunday**
- **Extra Office Hour TODAY! 5-6pm (H110)**
- **Sign up for peer tutoring**

  https://www.haverford.edu/academic-resources/peer-tutoring/find-tutor
- **CS dept Collaboration policy**

  https://docs.google.com/document/d/1o-V4qewRIyfhZe5S8BfjO_FRQ0FKMk8Ac3RCZ3rHrIg/edit

FEB 27 OUTLINE

- **Review check-in, recap Stacks**
- **Lab 3 suggestions**
- **Abstract Data Types (ADTs) and interfaces**
- **Implementing stacks**
- **Queues (theory and implementation)**

FEB 27 OUTLINE

- **Review check-in, recap Stacks**
- Lab 3 suggestions
- Abstract Data Types (ADTs) and interfaces
- Implementing stacks
- Queues (theory and implementation)

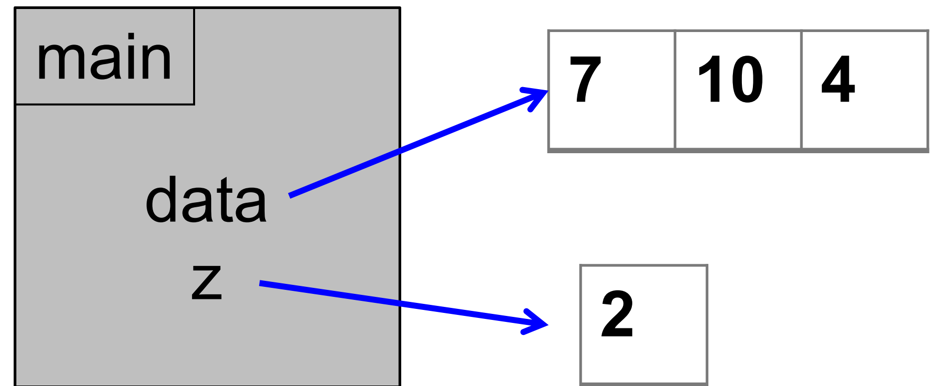
HANDOUT 12

```
1 def add(lst, x):
2     n = len(lst)
3     for i in range(n):
4         lst[i] = lst[i] + x
5     print("done adding!")
6
7 def main():
8     data = [7, 10, 4]
9     z = 2
10    add(data, z)
11    print(data)
12
13 main()
```

2. After the program above has finished, what is `data` equal to?

HANDOUT 12

```
1  def add(lst, x):
2      n = len(lst)
3      for i in range(n):
4          lst[i] = lst[i] + x
5      print("done adding!")
6
7  def main():
8      data = [7, 10, 4]
9      z = 2
10     add(data, z)
11     print(data)
12
13 main()
```

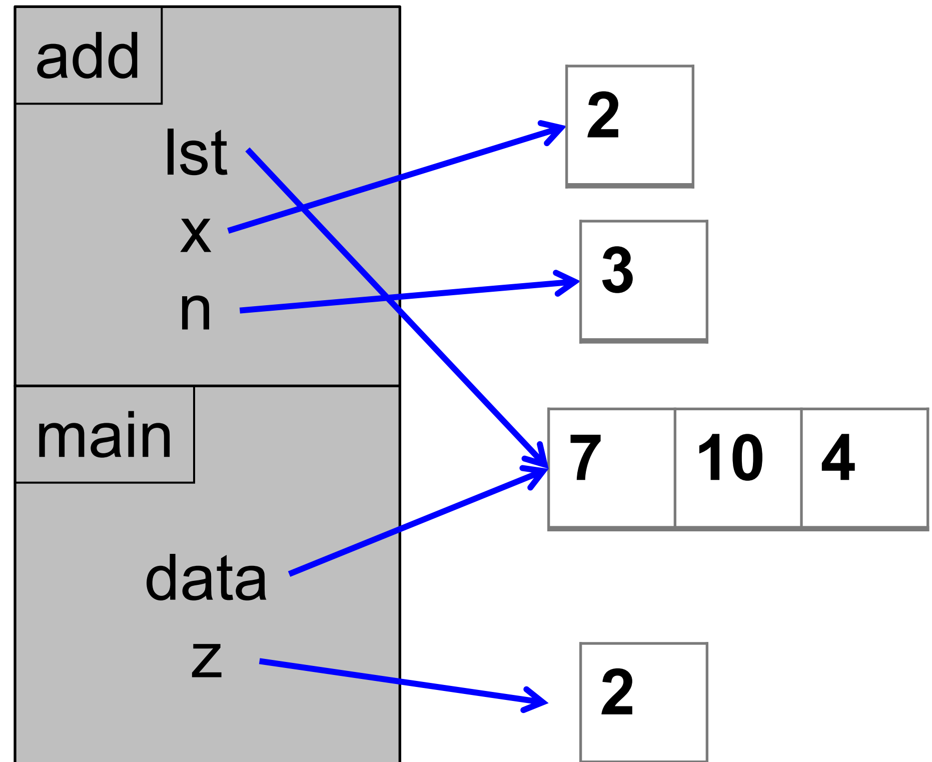


2. After the program above has finished, what is `data` equal to?

HANDOUT 12

Note: i omitted for simplicity

```
1 def add(lst, x):
2     n = len(lst)
3     for i in range(n):
4         lst[i] = lst[i] + x
5     print("done adding!")
6
7 def main():
8     data = [7, 10, 4]
9     z = 2
10    add(data, z)
11    print(data)
12
13 main()
```

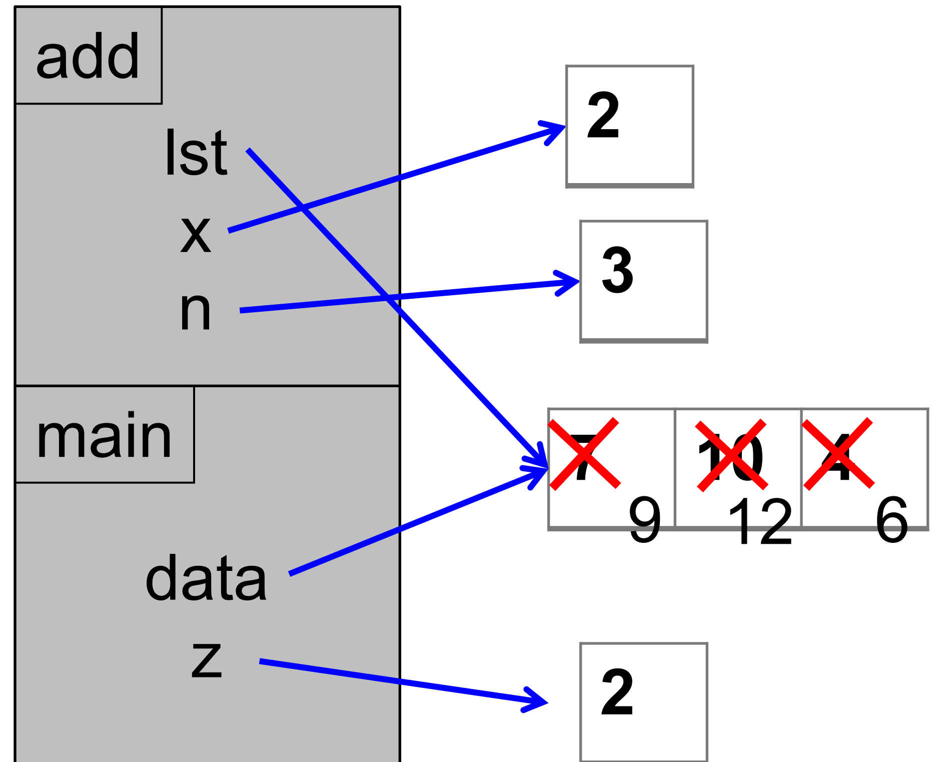


2. After the program above has finished, what is `data` equal to?

HANDOUT 12

Note: i omitted for simplicity

```
1  def add(lst, x):
2      n = len(lst)
3      for i in range(n):
4          lst[i] = lst[i] + x
5      print("done adding!")
6
7  def main():
8      data = [7, 10, 4]
9      z = 2
10     add(data, z)
11     print(data)
12
13 main()
```

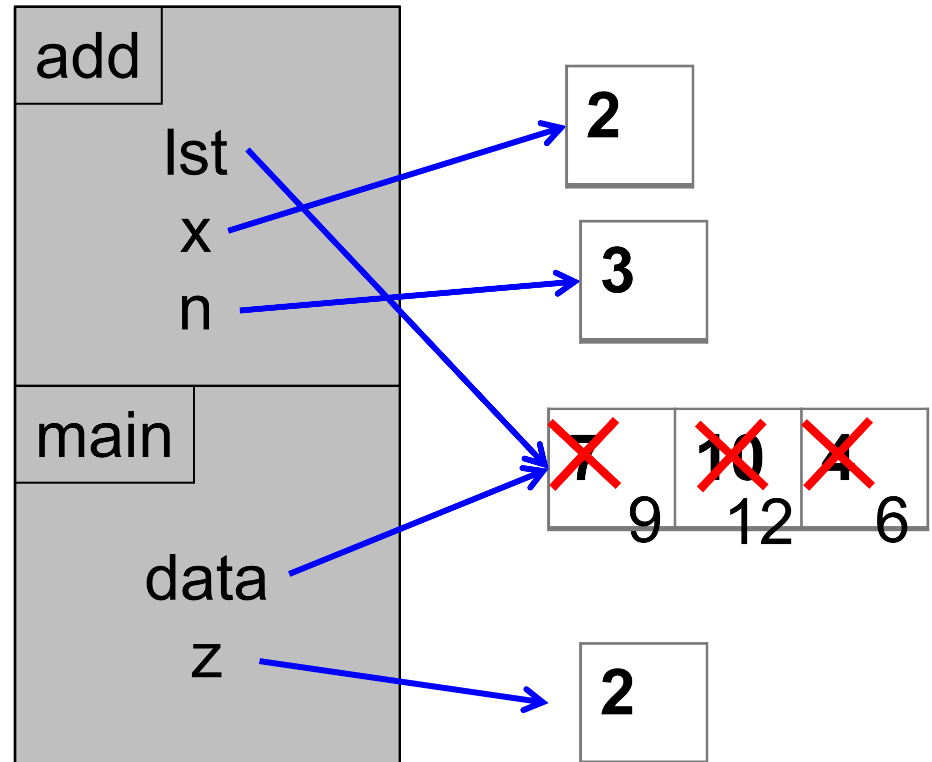


2. After the program above has finished, what is `data` equal to?

HANDOUT 12

Note: i omitted for simplicity

```
 1 def add(lst, x):
 2     n = len(lst)
 3     for i in range(n):
 4         lst[i] = lst[i] + x
 5     print("done adding!")
 6
 7 def main():
 8     data = [7, 10, 4]
 9     z = 2
10     add(data, z)
11     print(data)
12
13 main()
```



2. After the program above has finished, what is `data` equal to?

9	12	6

HANDOUT 12

3. Name at least two key methods that stacks must implement.

4. The runtime of all stack methods discussed on Tuesday is the same. What is it?

HANDOUT 12

3. Name at least two key methods that stacks must implement.

```
void push(E element)     // add element to top of stack
E pop()                  // remove and return top
E peek()                 // view top but don't remove
boolean isEmpty()        // check if empty
int size()               // number of elements
```

4. The runtime of all stack methods discussed on Tuesday is the same. What is it?

HANDOUT 12

3. Name at least two key methods that stacks must implement.

```
void push(E element)     // add element to top of stack
E pop()                  // remove and return top
E peek()                 // view top but don't remove
boolean isEmpty()        // check if empty
int size()               // number of elements
```

4. The runtime of all stack methods discussed on Tuesday is the same. What is it?

Constant! O(1)
Stacks are fast by limiting functionality

HANDOUT 12

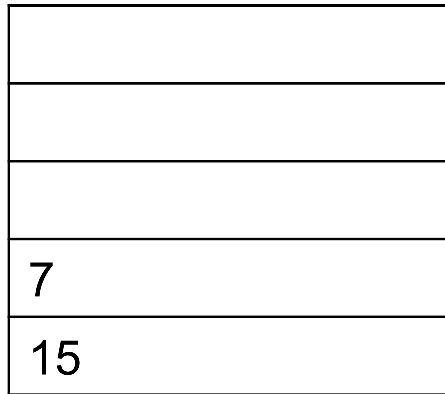
5. For the postfix expression below, use a stack to compute the value (show steps).

$15 \ \ 7 - 2 \times 4 \ /$

HANDOUT 12

5. For the postfix expression below, use a stack to compute the value (show steps).

$15 \;\; 7 - 2 \times 4 \; /$



HANDOUT 12

5. For the postfix expression below, use a stack to compute the value (show steps).

$15 \ 7 - 2 \times 4 \ /$

~~7~~ 2
~~15~~ 8

stack

15 – 7 = 8

HANDOUT 12

5. For the postfix expression below, use a stack to compute the value (show steps).

$15 \ \ 7 - 2 \times 4 \ /$

~~7~~ ~~2~~ 4
~~15~~ ~~8~~ 16

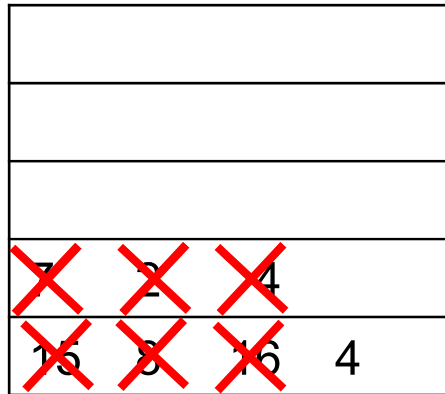
stack

15 – 7 = 8
8 * 2 = 16

HANDOUT 12

5. For the postfix expression below, use a stack to compute the value (show steps).

$$15 \;\; 7 - 2 \times 4 \; /$$




```
15 – 7 = 8
8 * 2 = 16
16 / 4 = 4

return: 4
```

HANDOUT 12

6. *Extension question*: if we were to implement a stack using an array, what instance variables would you store? What are the advantages/disadvantages of an array-based stack?

HANDOUT 12

6. *Extension question*: if we were to implement a stack using an array, what instance variables would you store? What are the advantages/disadvantages of an array-based stack?

Array containing the data
Integer representing the top of the stack

Difficult to add more elements once the stack is full!

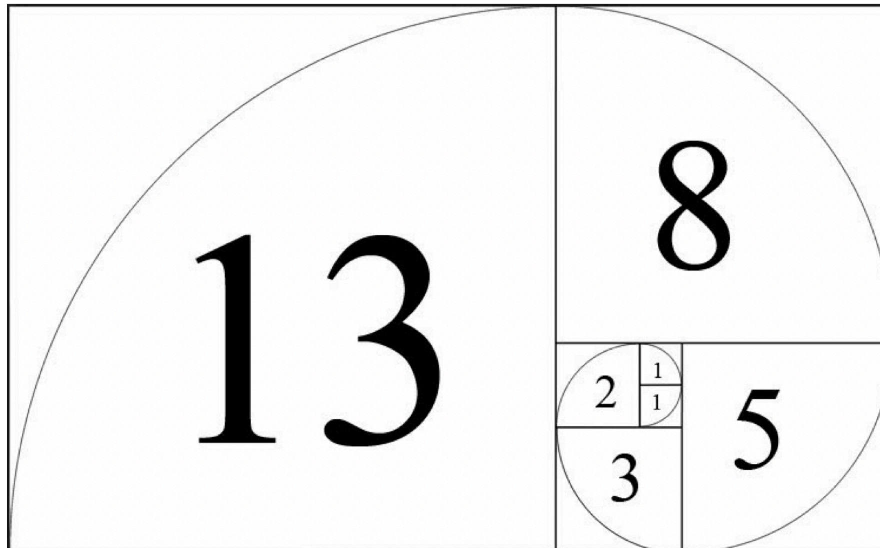
Fibonacci Stack Example

FIBONACCI NUMBERS

Each Fibonacci number is the sum of the previous two Fibonacci numbers

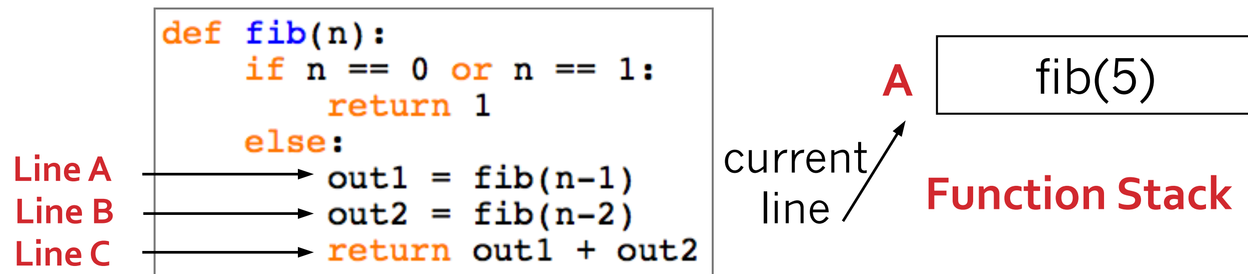
Recursion: $F_n = F_{n-1} + F_{n-2}$

Base cases: $F_0 = 1$ and $F_1 = 1$

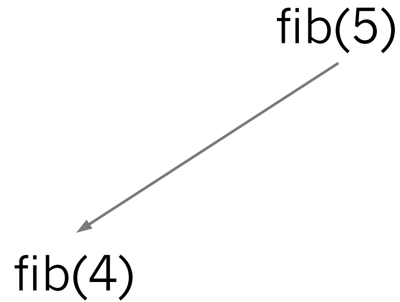


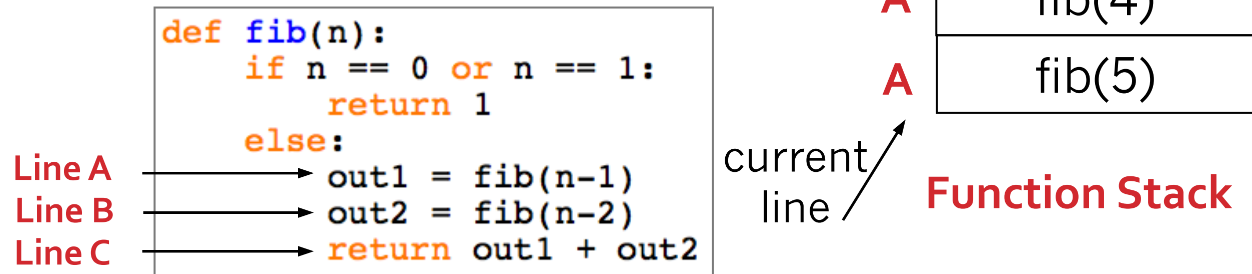
FIBONACCI FUNCTION STACK

fib(5)



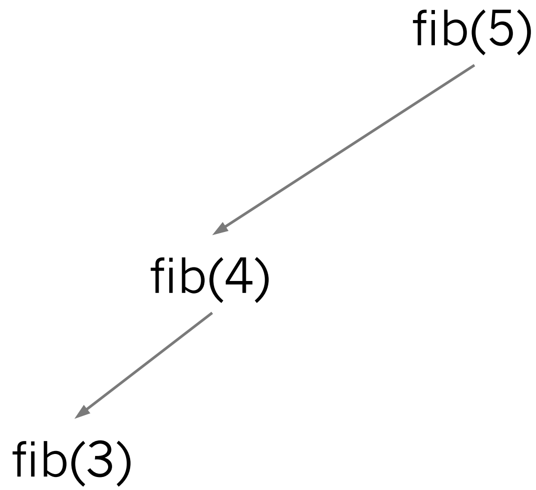
FIBONACCI FUNCTION STACK





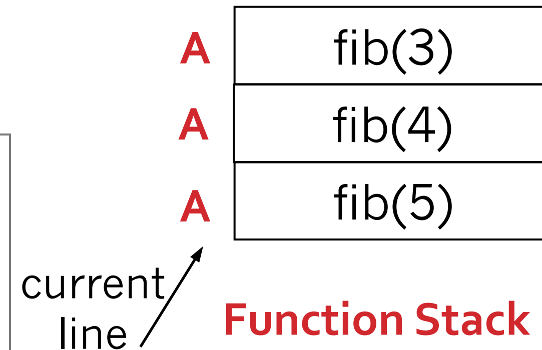

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

FIBONACCI FUNCTION STACK

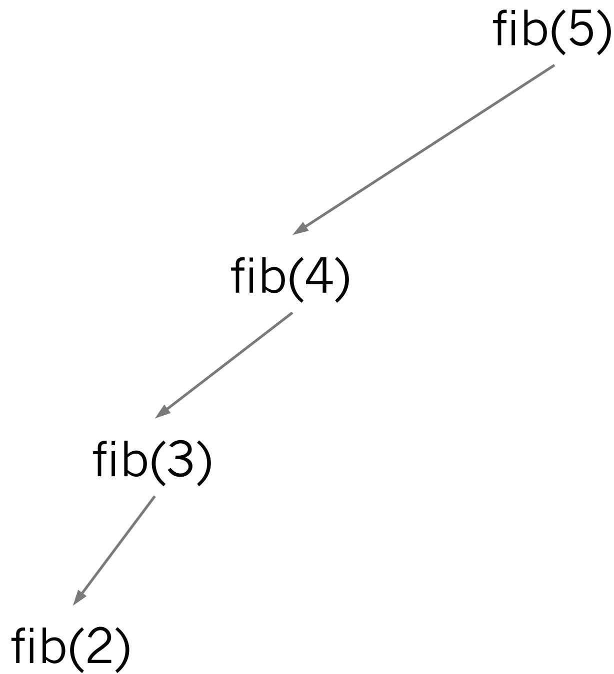


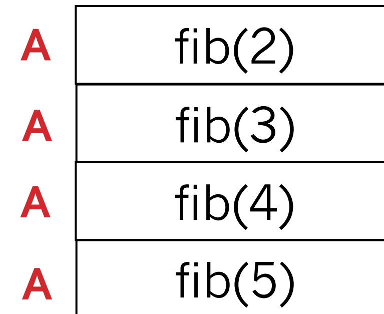
Line A → `out1 = fib(n-1)`
Line B → `out2 = fib(n-2)`
Line C → `return out1 + out2`

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



FIBONACCI FUNCTION STACK



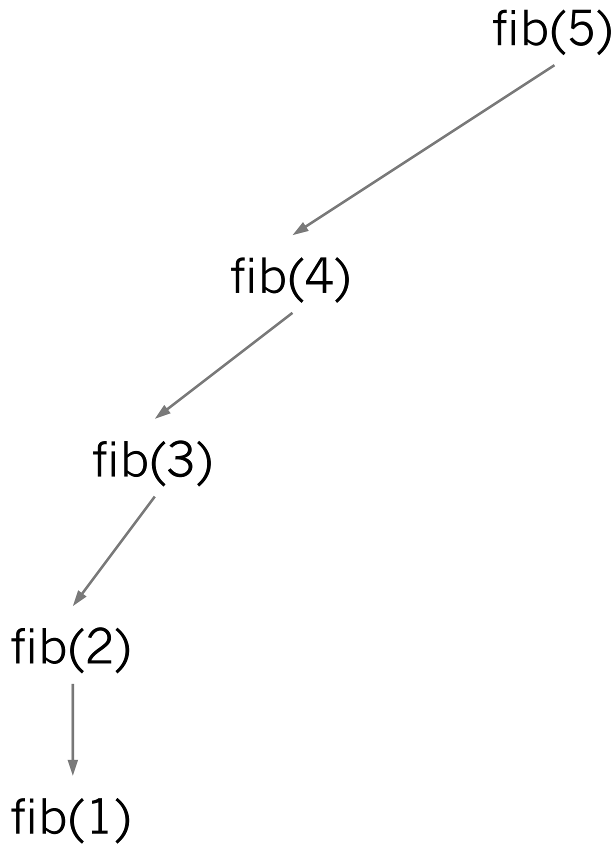


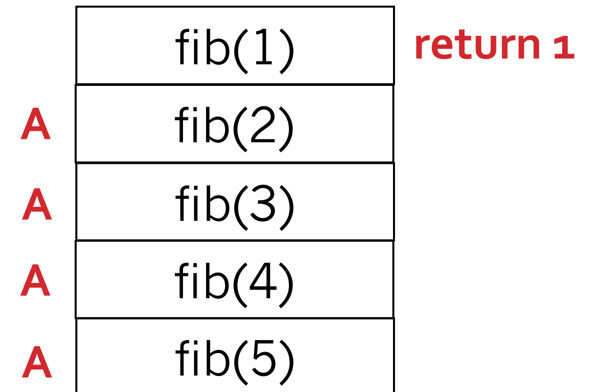
current line

Function Stack

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

FIBONACCI FUNCTION STACK





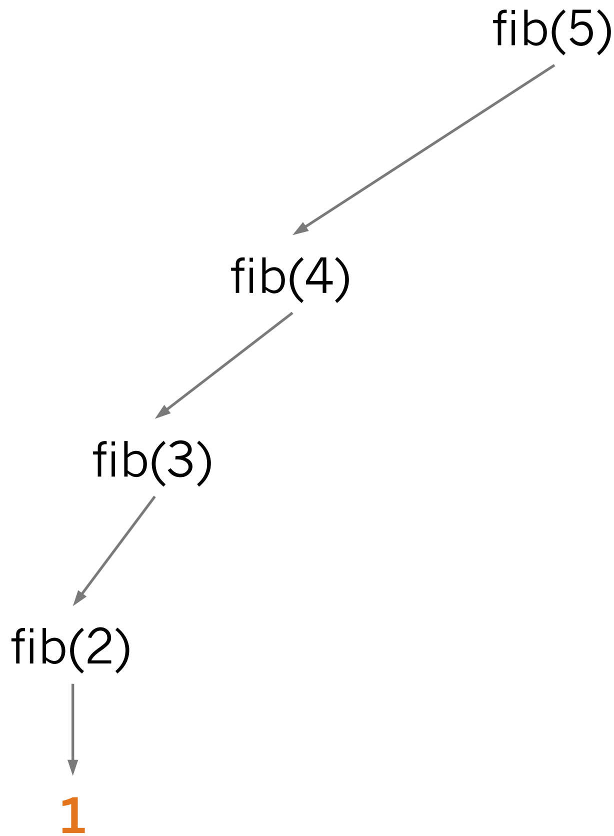

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A → out1 = fib(n-1)
Line B → out2 = fib(n-2)
Line C → return out1 + out2

current line

Function Stack

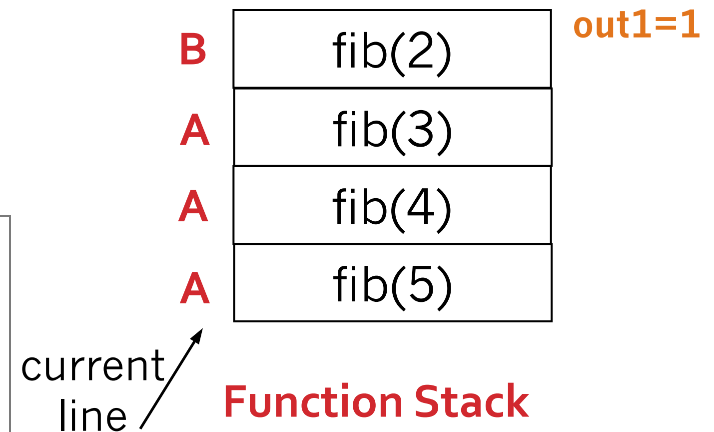
FIBONACCI FUNCTION STACK




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

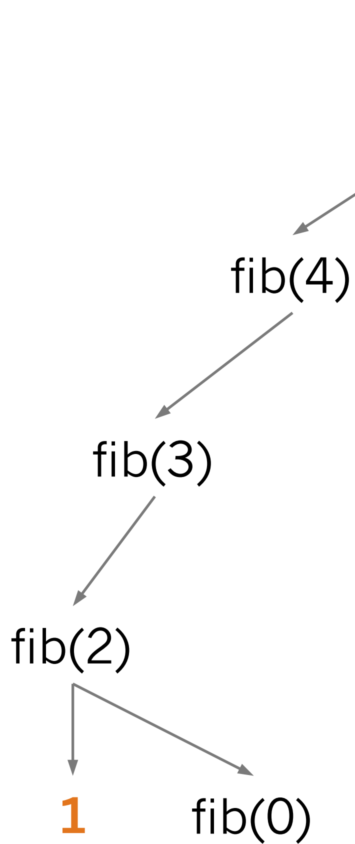
Line A
Line B
Line C

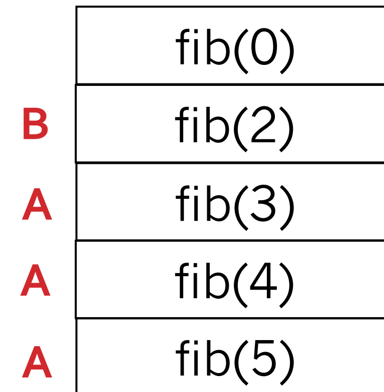
current line



FIBONACCI FUNCTION STACK

fib(5)





return 1
out1=1

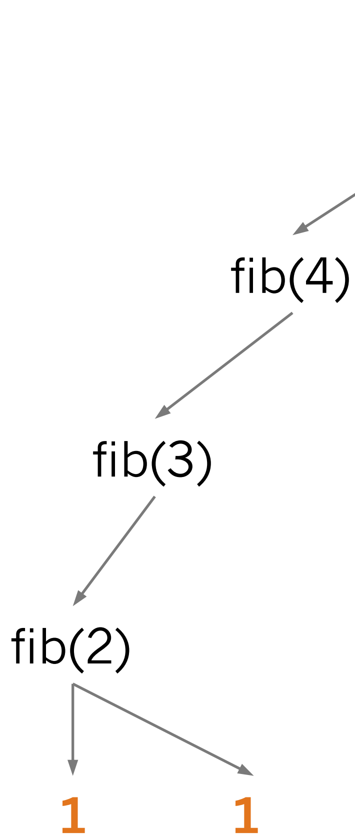
current line

Function Stack

```
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 → out1 = fib(n-1)
Line B → out2 = fib(n-2)
Line C → return out1 + out2

FIBONACCI FUNCTION STACK



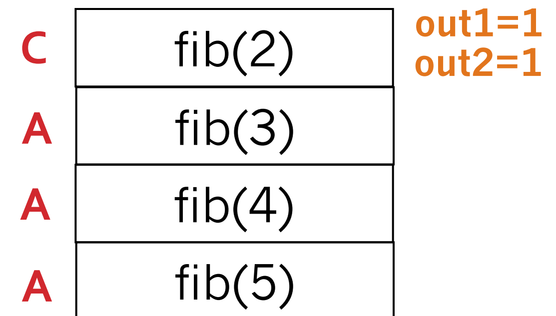

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A → `out1 = fib(n-1)`

Line B → `out2 = fib(n-2)`

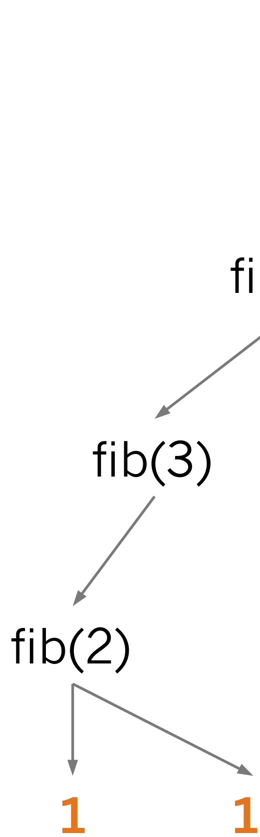
Line C → `return out1 + out2`

current line



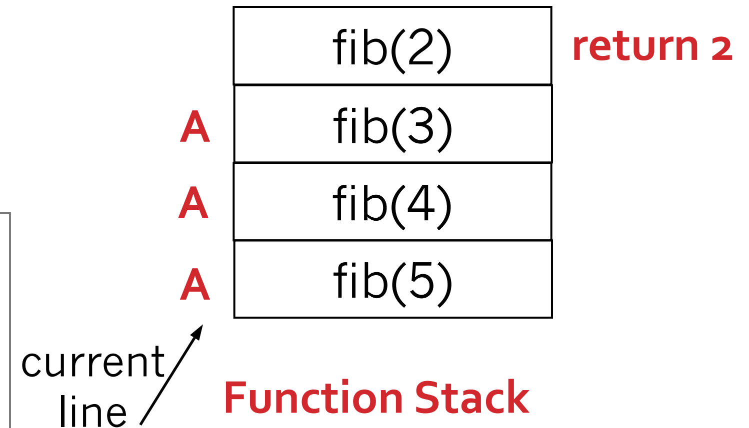
Function Stack

FIBONACCI FUNCTION STACK

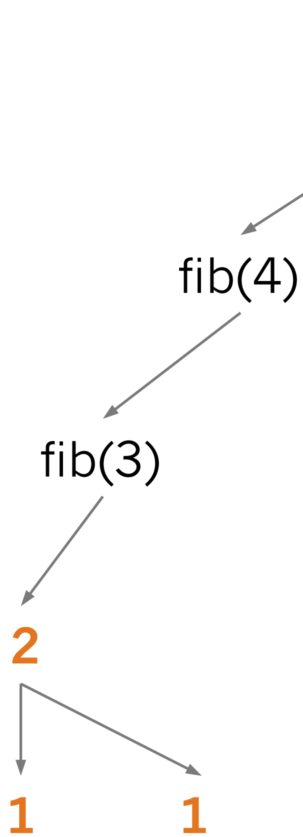



```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A
Line B
Line C

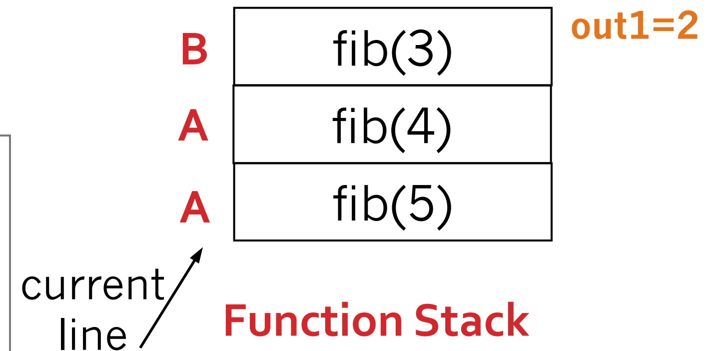


FIBONACCI FUNCTION STACK

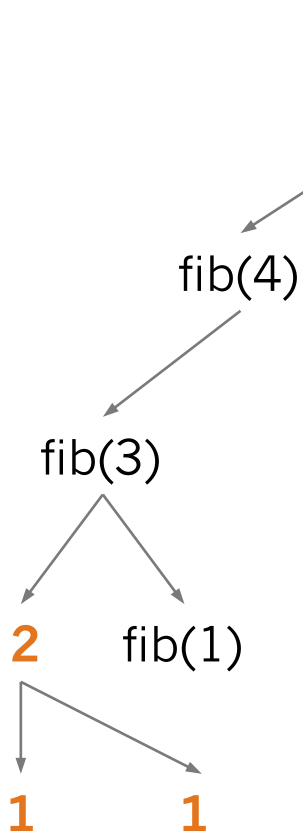



```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A → out1 = fib(n-1)
Line B → out2 = fib(n-2)
Line C → return out1 + out2

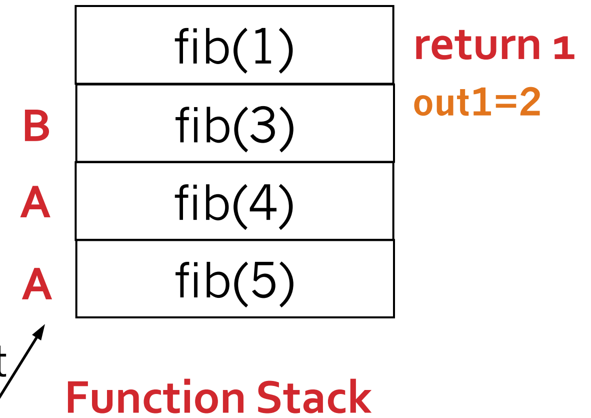


FIBONACCI FUNCTION STACK

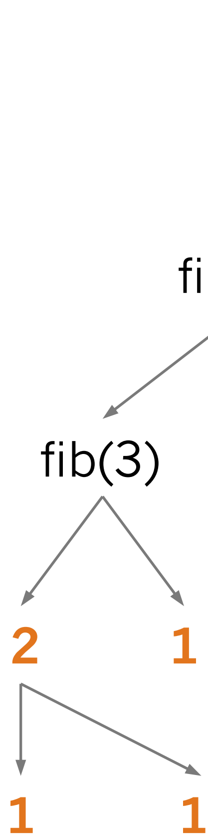



```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A → out1 = fib(n-1)
Line B → out2 = fib(n-2)
Line C → return out1 + out2

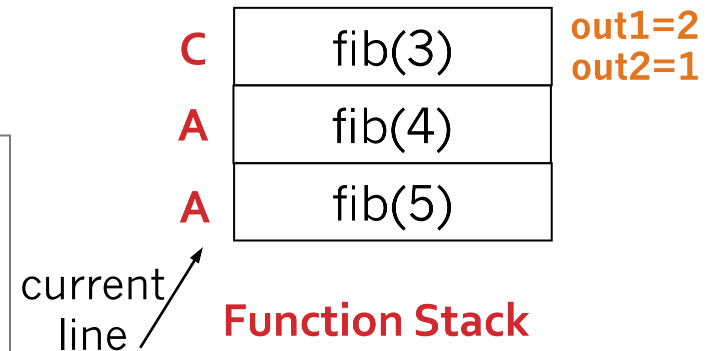


FIBONACCI FUNCTION STACK



Line A
Line B
Line C

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

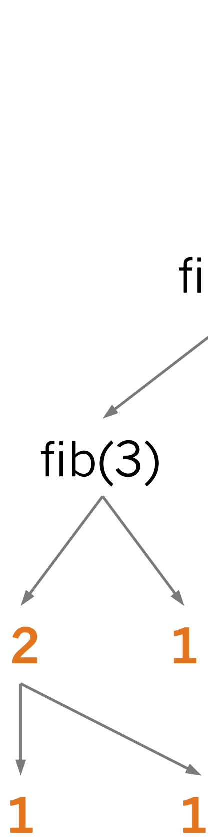


FIBONACCI FUNCTION STACK

fib(5)

fib(4)

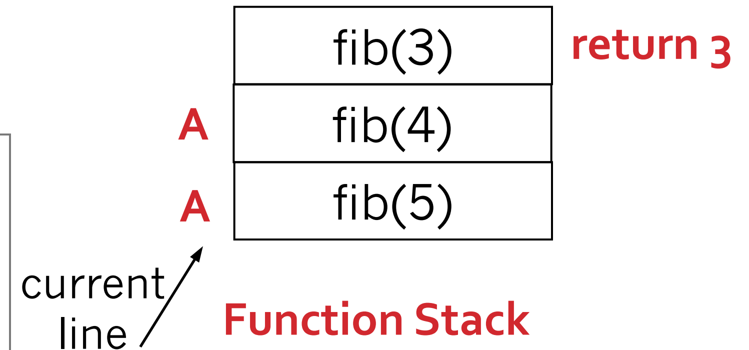
fib(3)




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

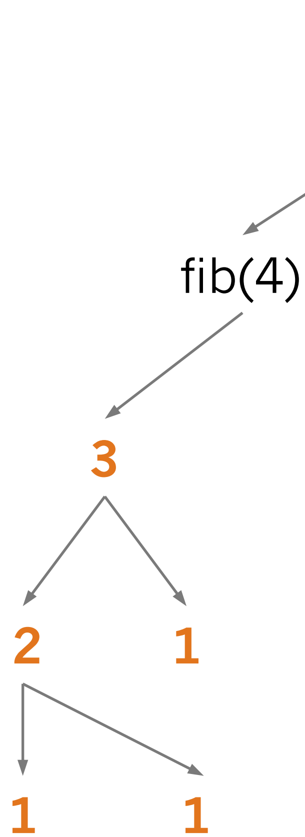
Line A
Line B
Line C

current line



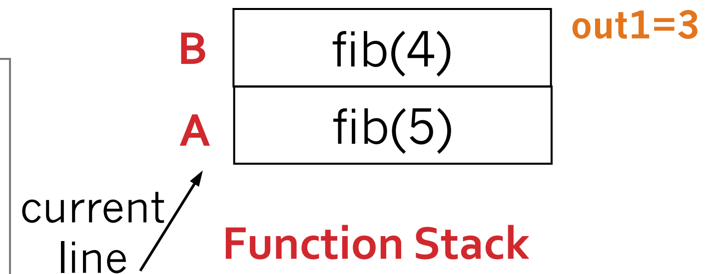
FIBONACCI FUNCTION STACK

fib(5)

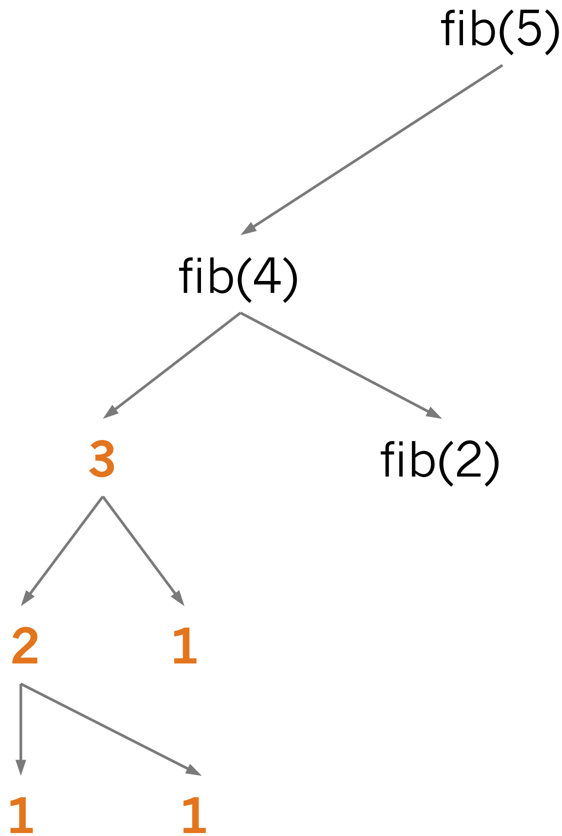



```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A → `out1 = fib(n-1)`
Line B → `out2 = fib(n-2)`
Line C → `return out1 + out2`



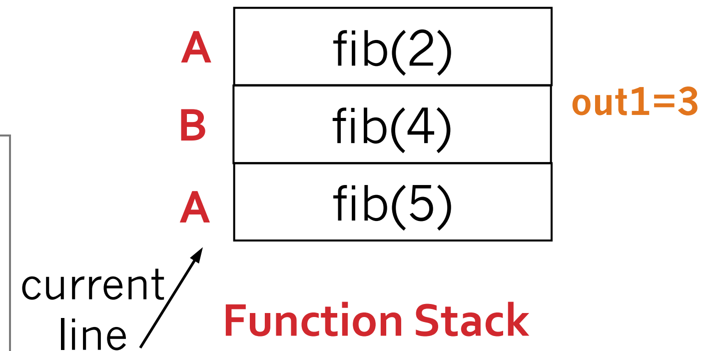
FIBONACCI FUNCTION STACK



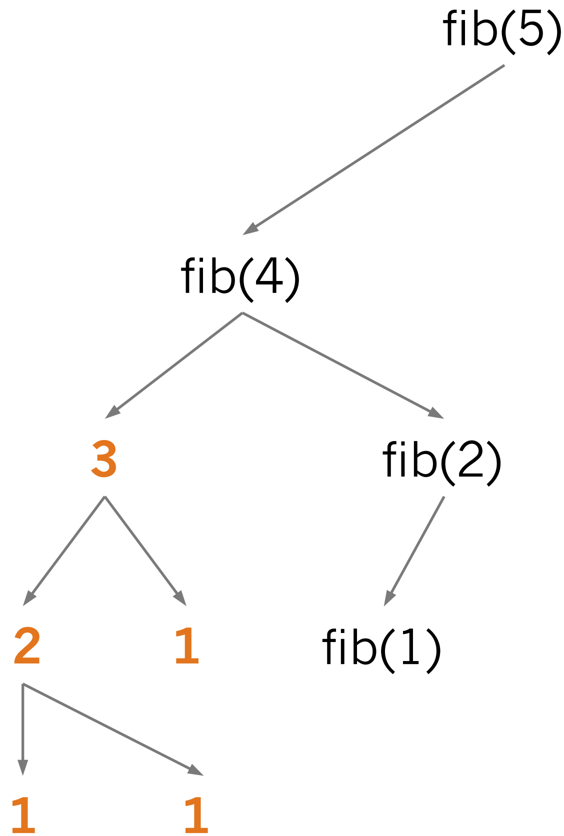

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A → out1 = fib(n-1)
Line B → out2 = fib(n-2)
Line C → return out1 + out2

current line

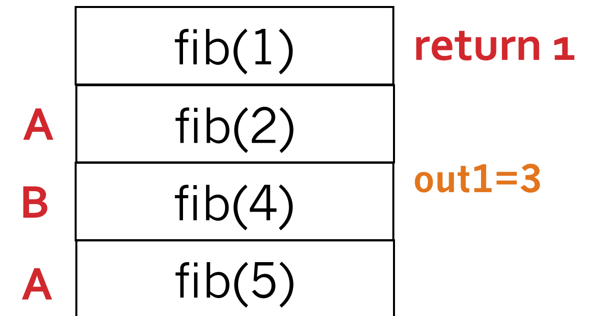


FIBONACCI FUNCTION STACK



Line A
Line B
Line C

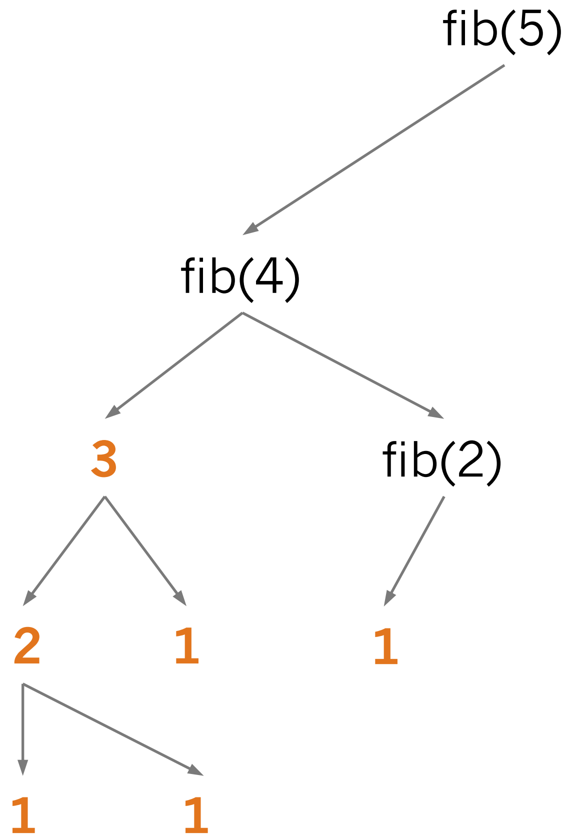
```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```



current line

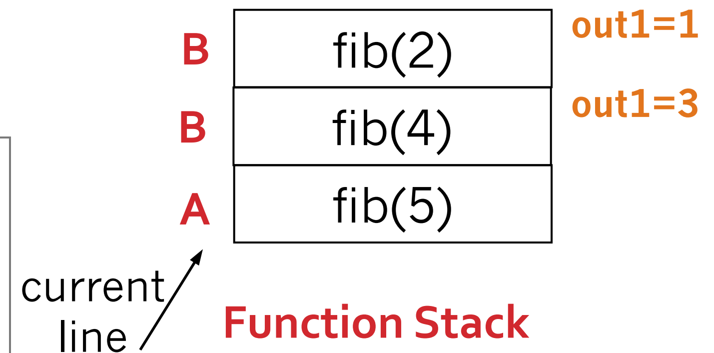
Function Stack

FIBONACCI FUNCTION STACK

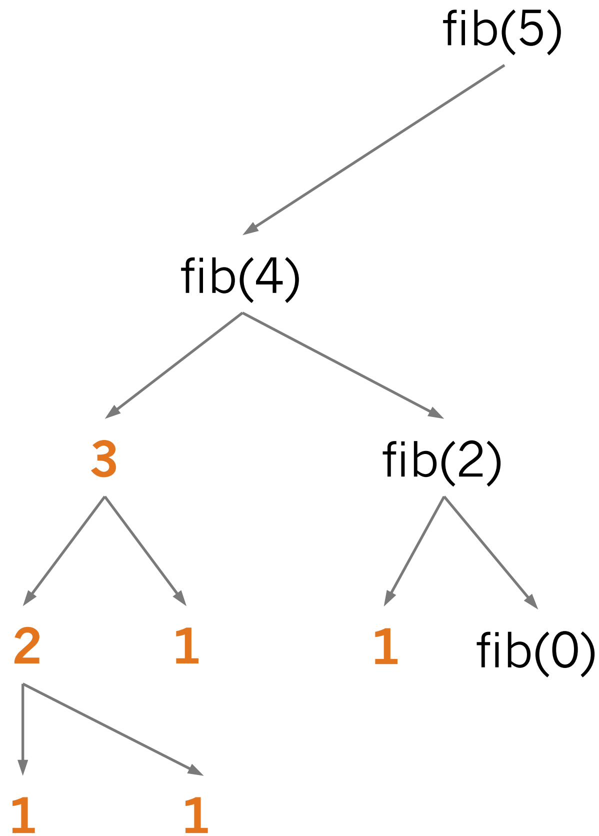


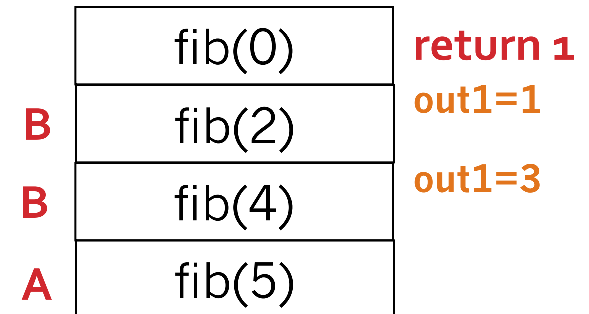
Line A
Line B
Line C

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



FIBONACCI FUNCTION STACK





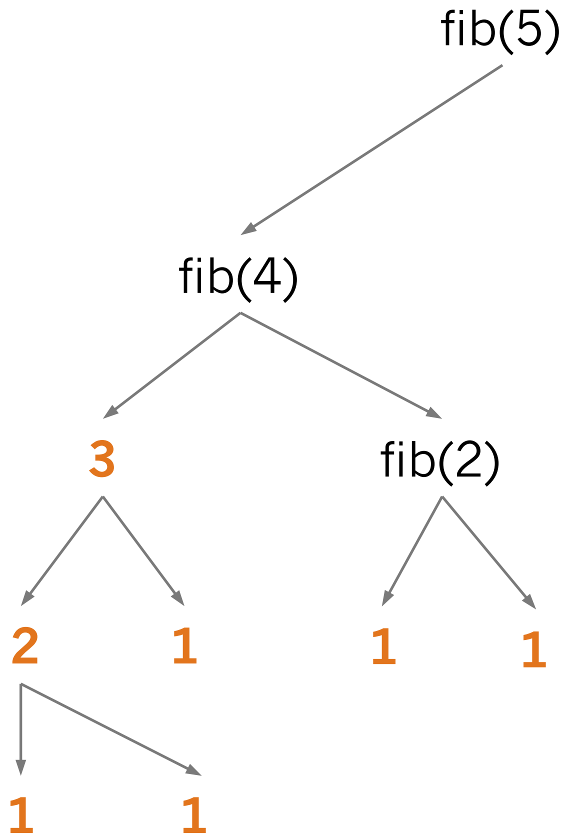
Function Stack

current line

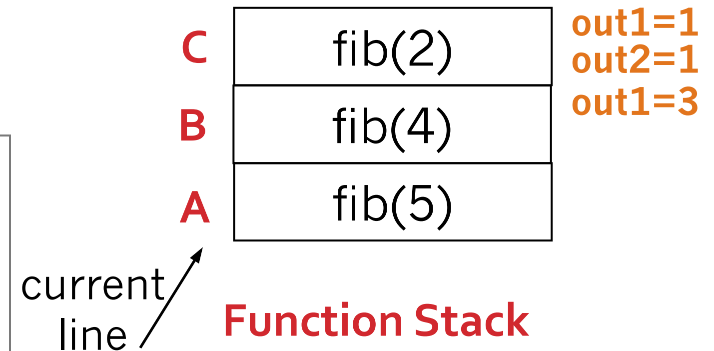
Line A
Line B
Line C

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

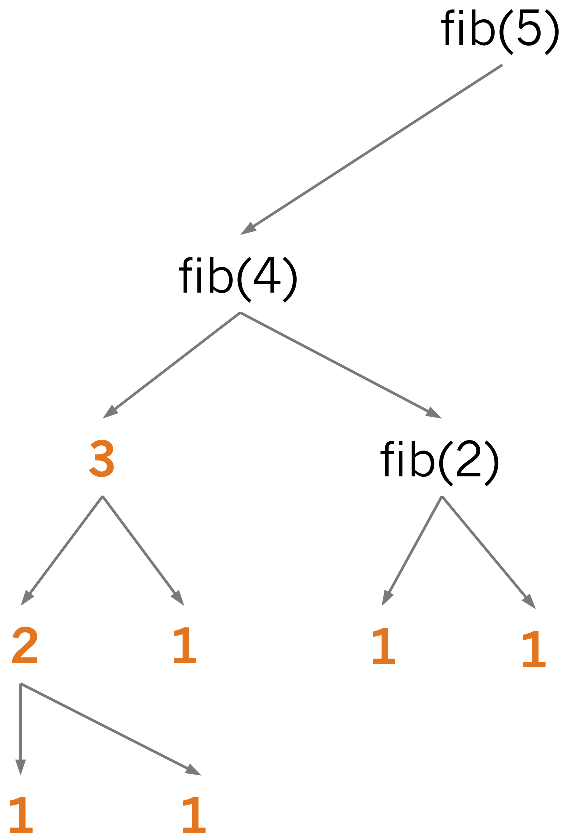
FIBONACCI FUNCTION STACK




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



FIBONACCI FUNCTION STACK

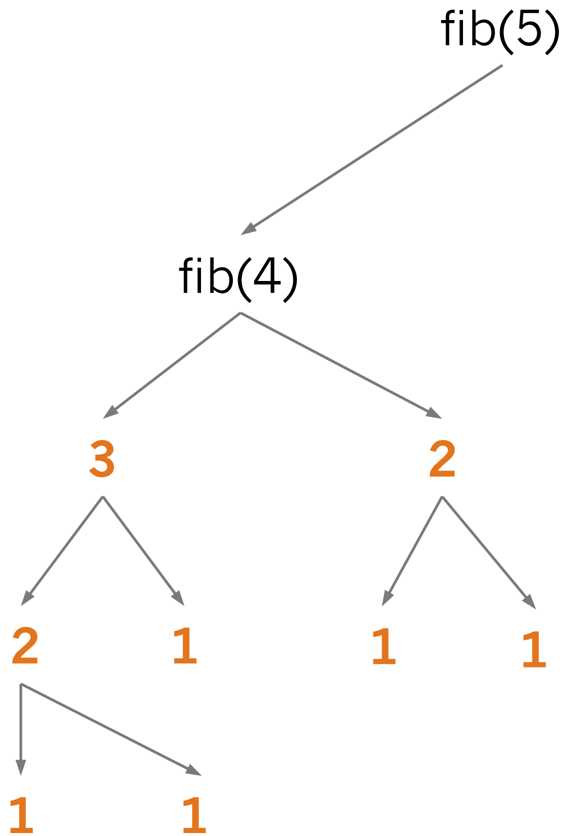


Line A
Line B
Line C

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

current line

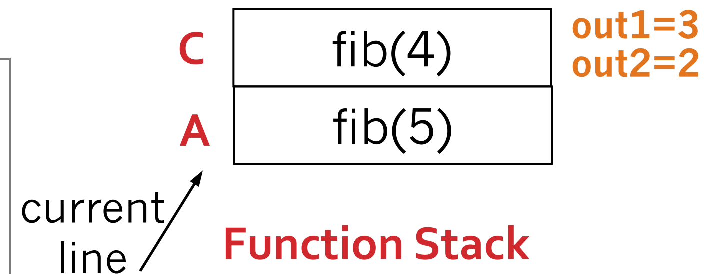
FIBONACCI FUNCTION STACK



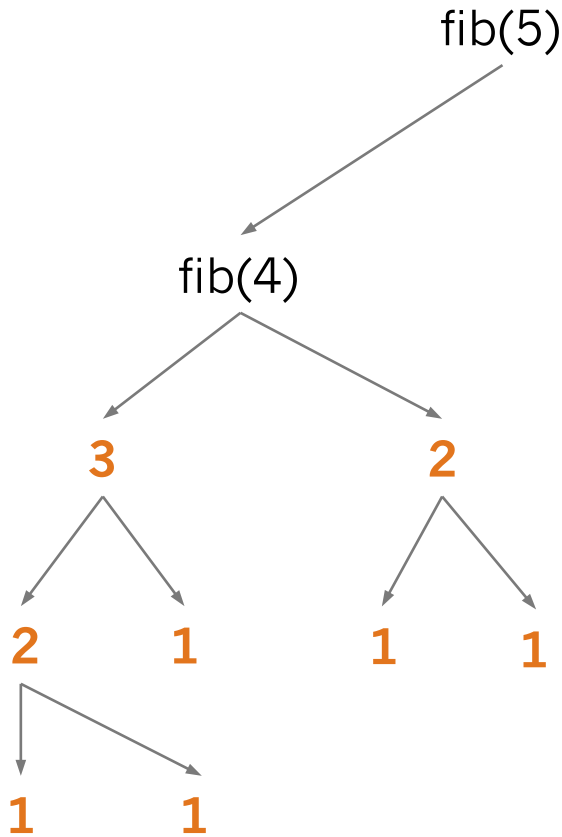

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A → out1 = fib(n-1)
Line B → out2 = fib(n-2)
Line C → return out1 + out2

current line

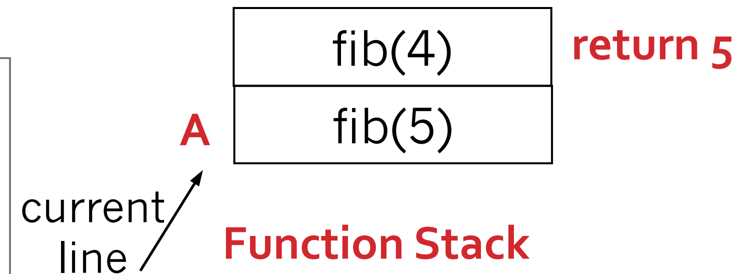


FIBONACCI FUNCTION STACK

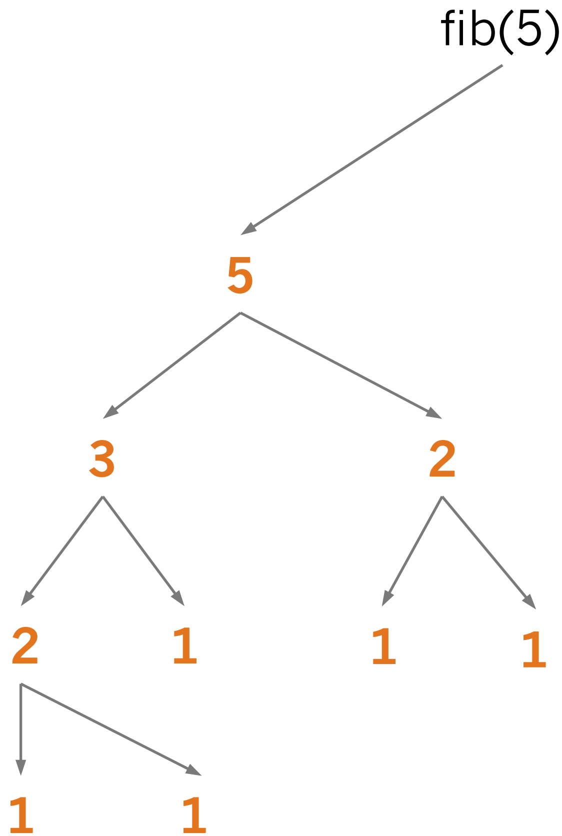



```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A → out1 = fib(n-1)
Line B → out2 = fib(n-2)
Line C → return out1 + out2

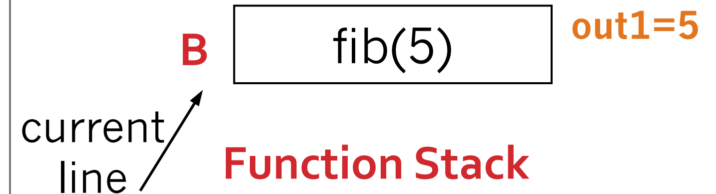


FIBONACCI FUNCTION STACK

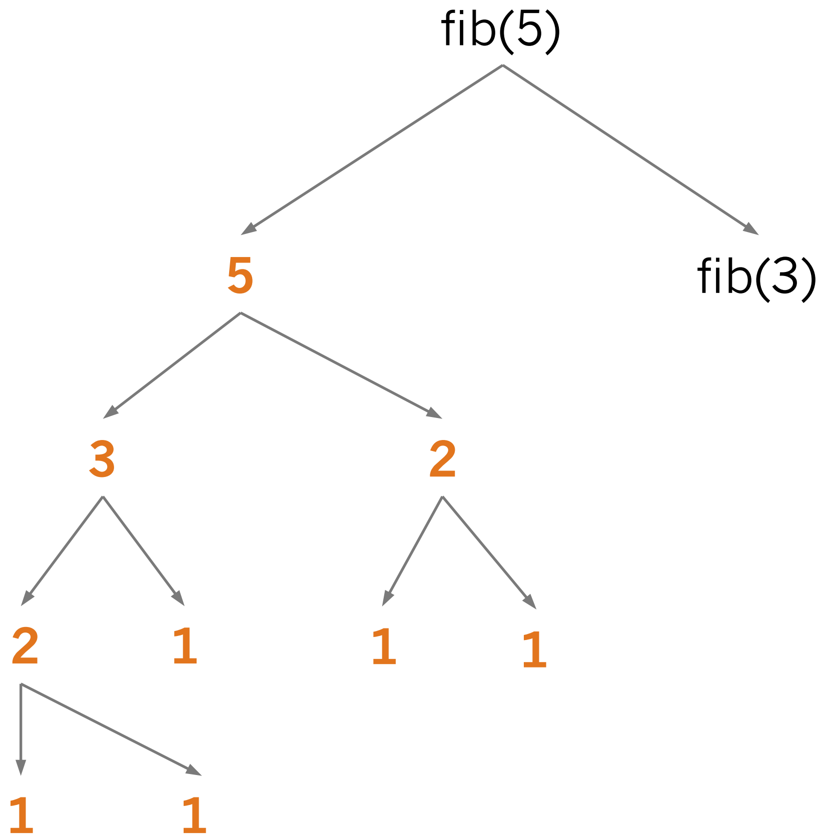



```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A → `out1 = fib(n-1)`
Line B → `out2 = fib(n-2)`
Line C → `return out1 + out2`

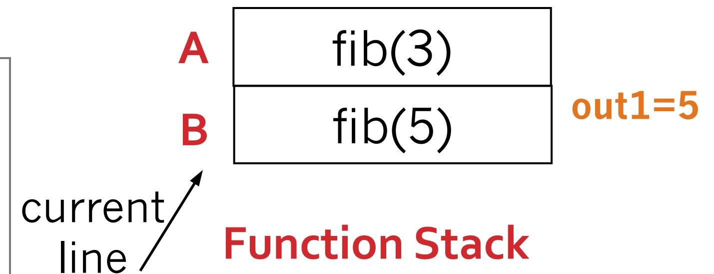


FIBONACCI FUNCTION STACK

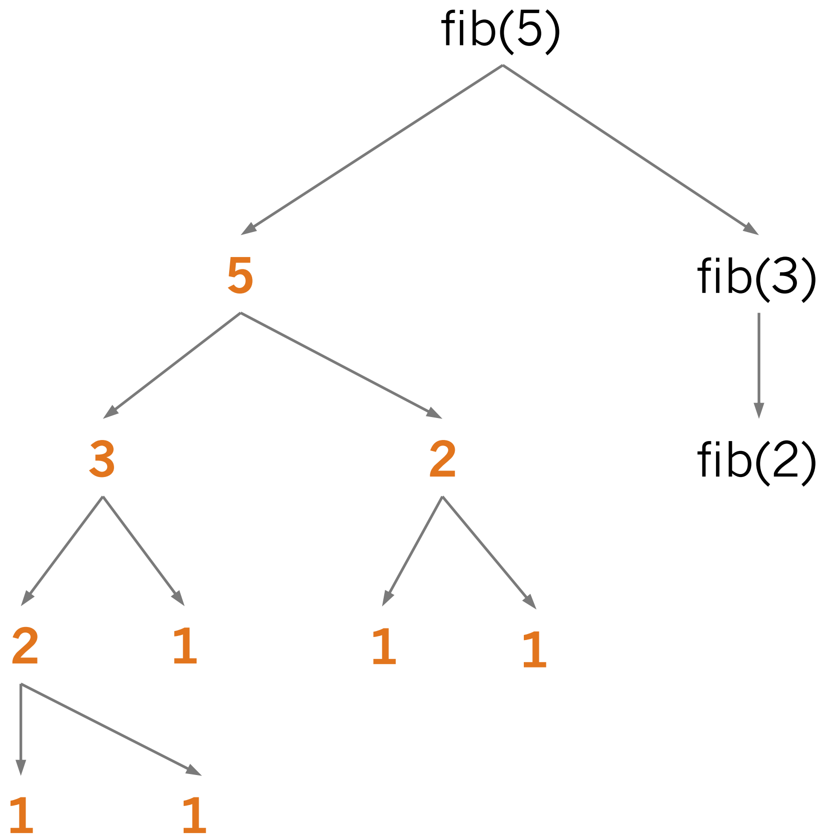


Line A
Line B
Line C

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



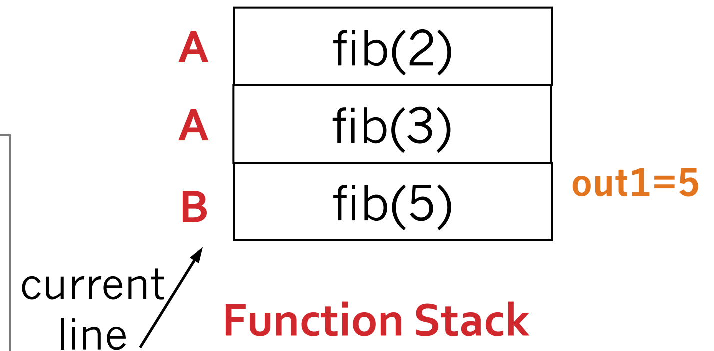
FIBONACCI FUNCTION STACK



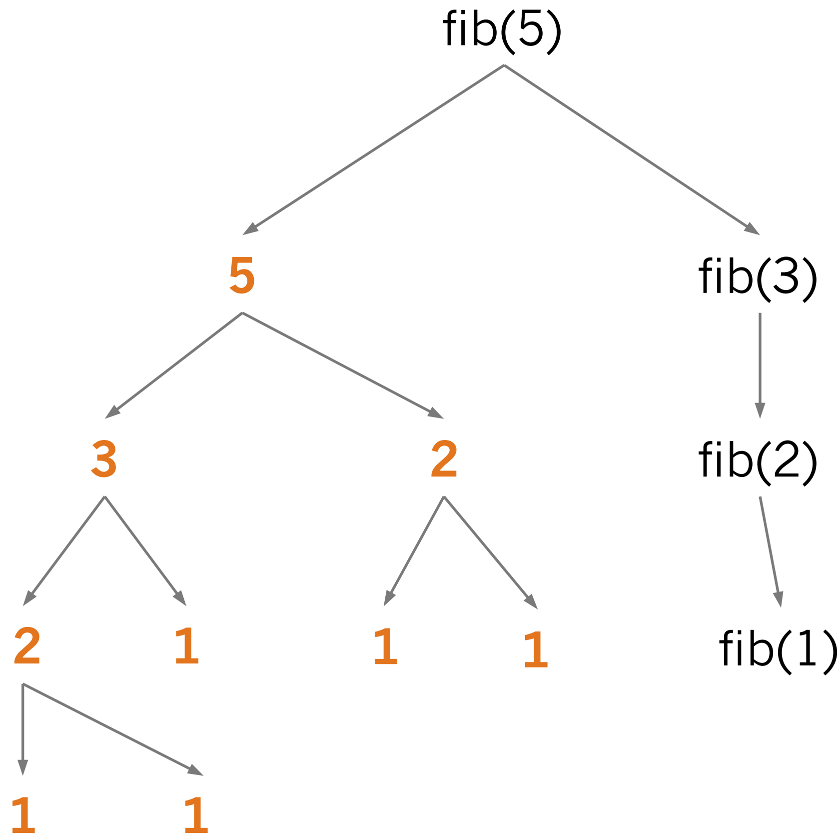

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A → out1 = fib(n-1)
Line B → out2 = fib(n-2)
Line C → return out1 + out2

current line



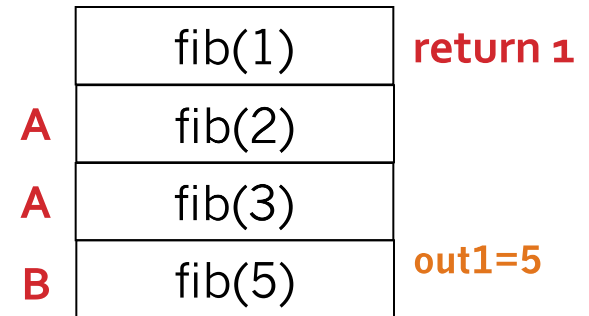
FIBONACCI FUNCTION STACK



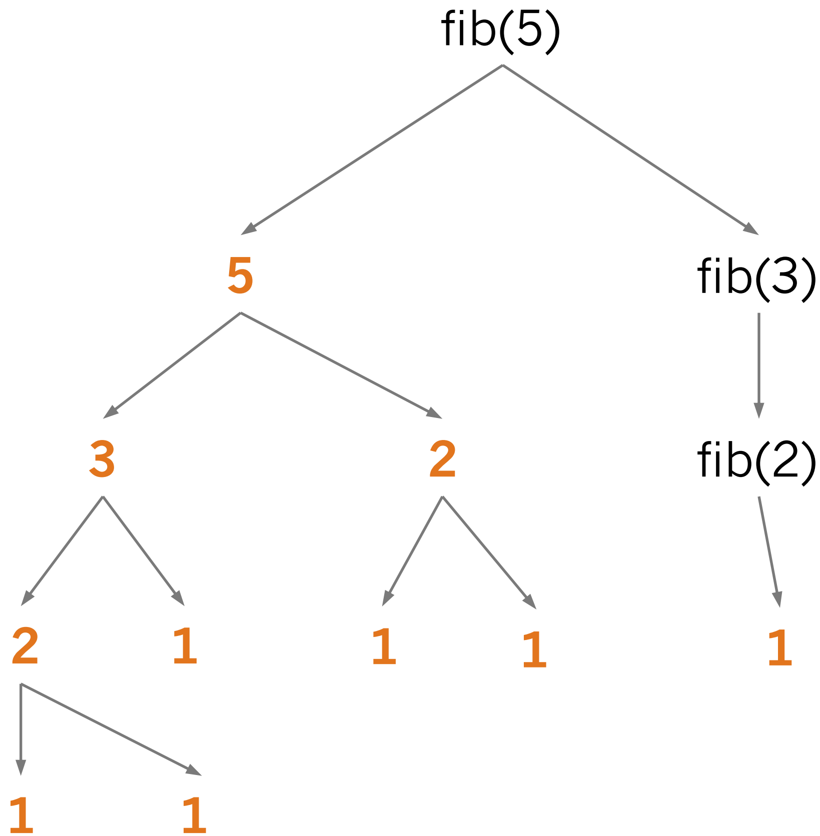

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

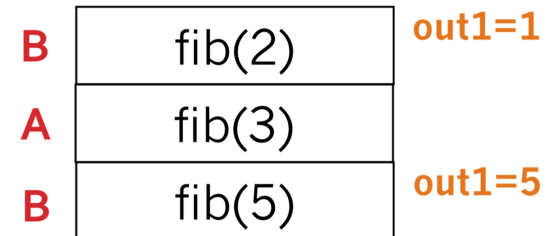
Line A → `out1 = fib(n-1)`
Line B → `out2 = fib(n-2)`
Line C → `return out1 + out2`

current line



FIBONACCI FUNCTION STACK





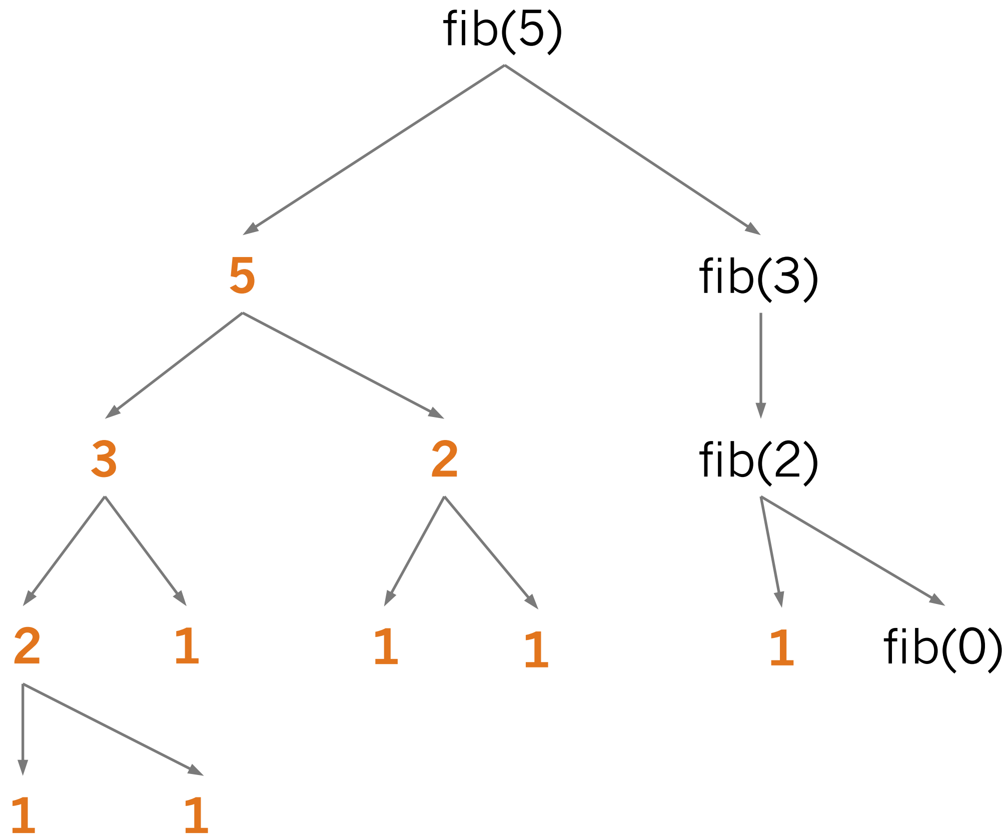
Line A
Line B
Line C

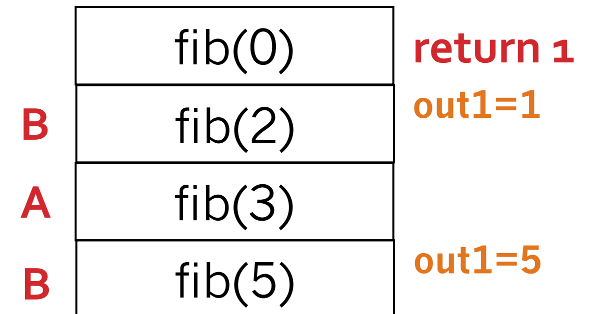
```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

current line

Function Stack

FIBONACCI FUNCTION STACK






```
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 → out1 = fib(n-1)

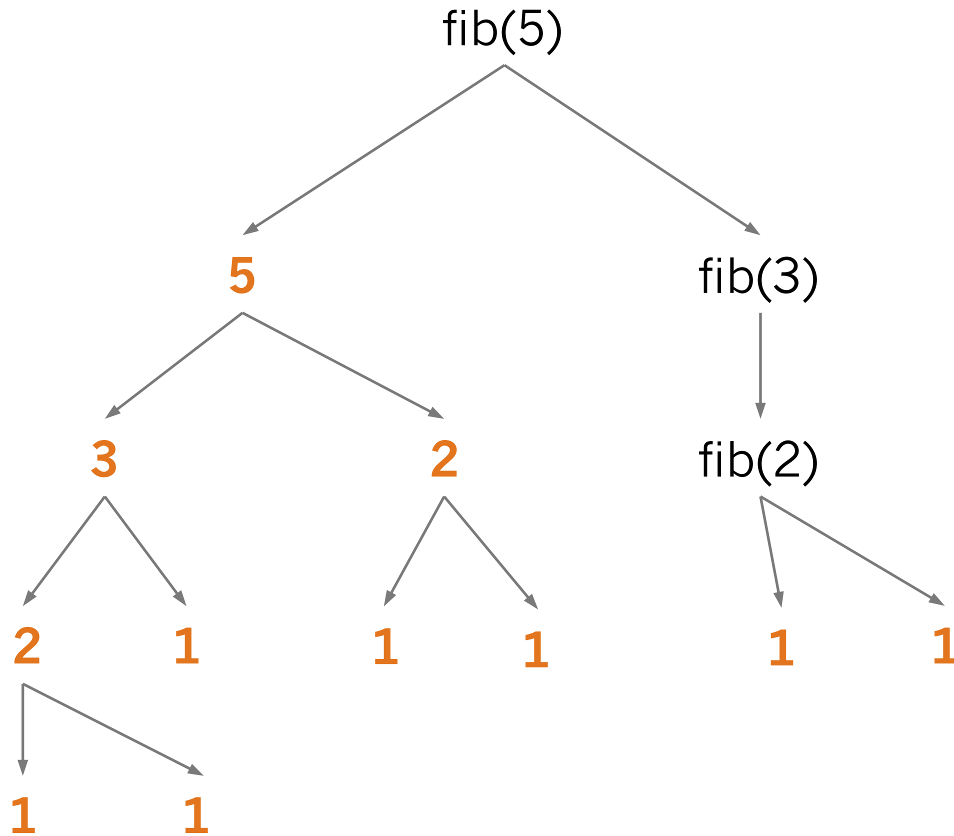
Line B → out2 = fib(n-2)

Line C → return out1 + out2

current line

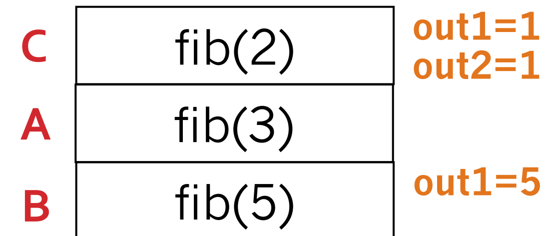
Function Stack

FIBONACCI FUNCTION STACK



Line A → `out1 = fib(n-1)`
Line B → `out2 = fib(n-2)`
Line C → `return out1 + out2`

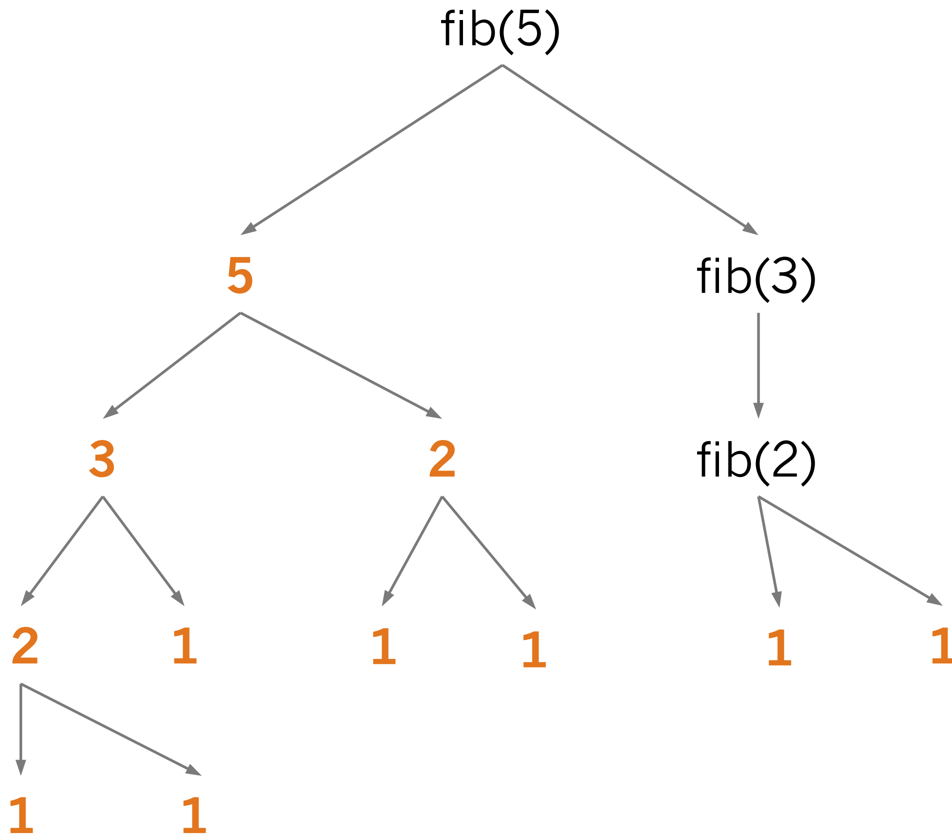
```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```



current line

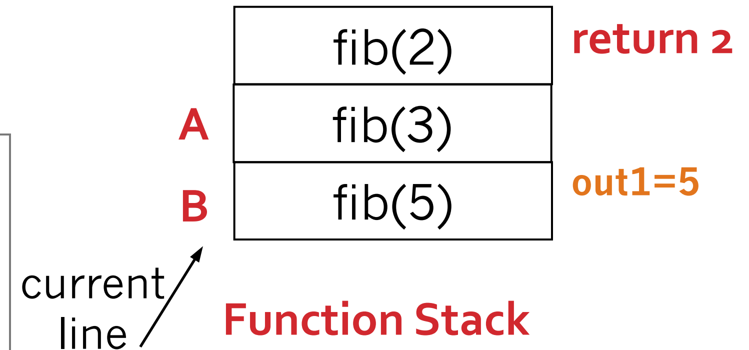
Function Stack

FIBONACCI FUNCTION STACK

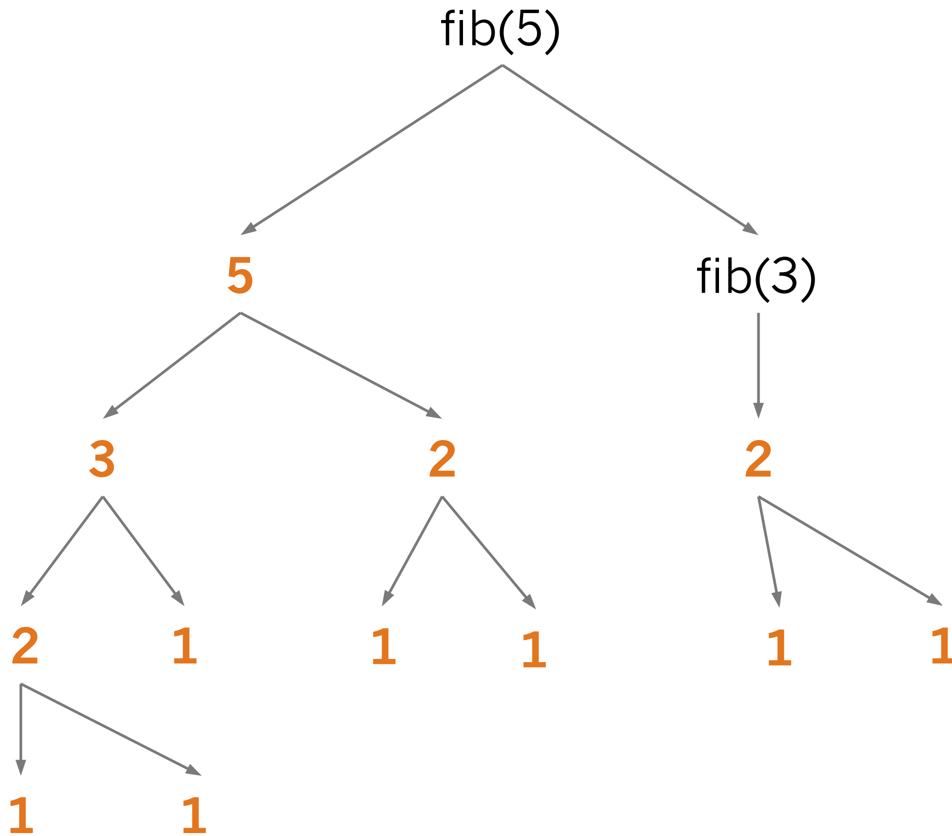



```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A
Line B
Line C



FIBONACCI FUNCTION STACK

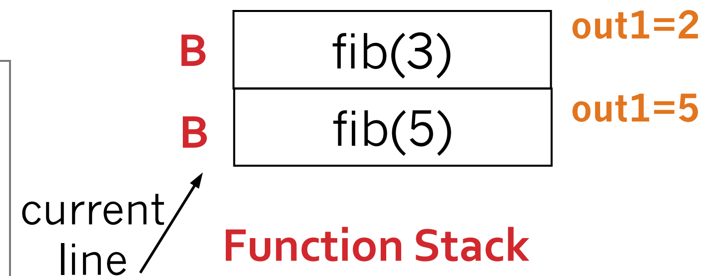



```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

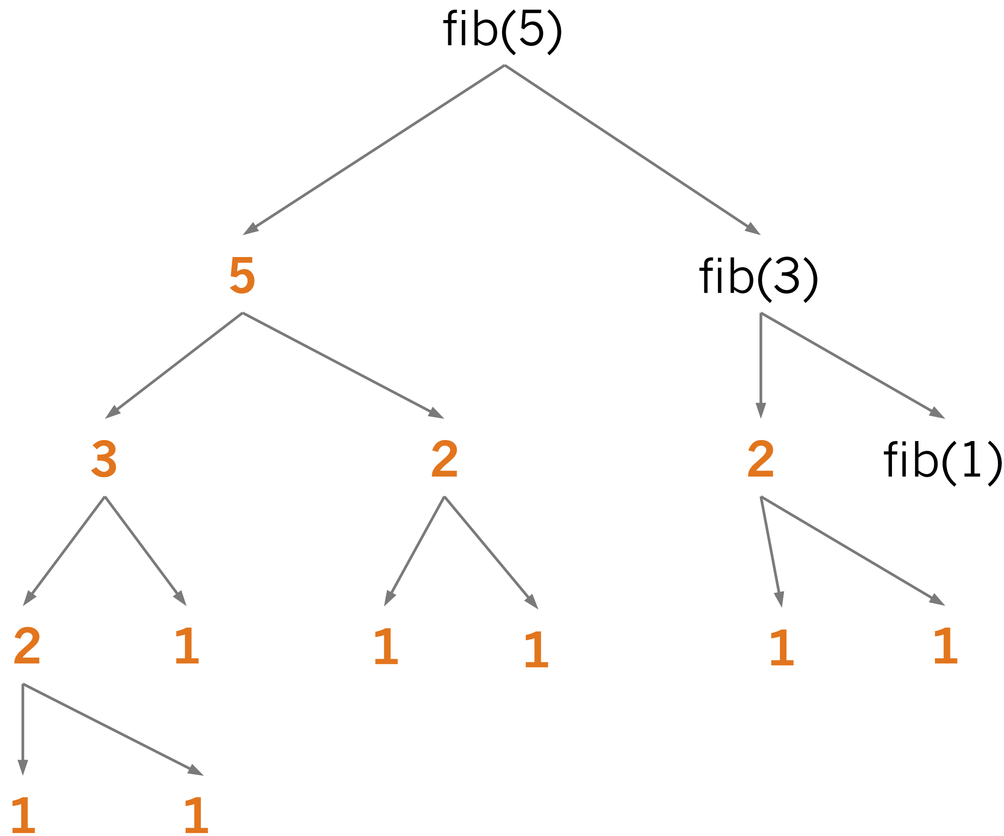
Line A → `out1 = fib(n-1)`

Line B → `out2 = fib(n-2)`

Line C → `return out1 + out2`

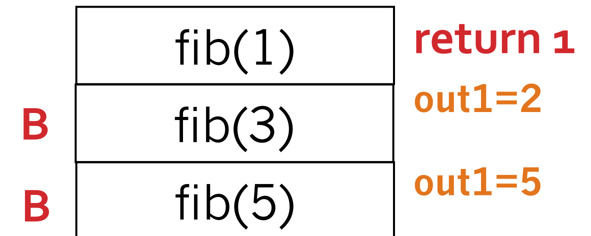


FIBONACCI FUNCTION STACK




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

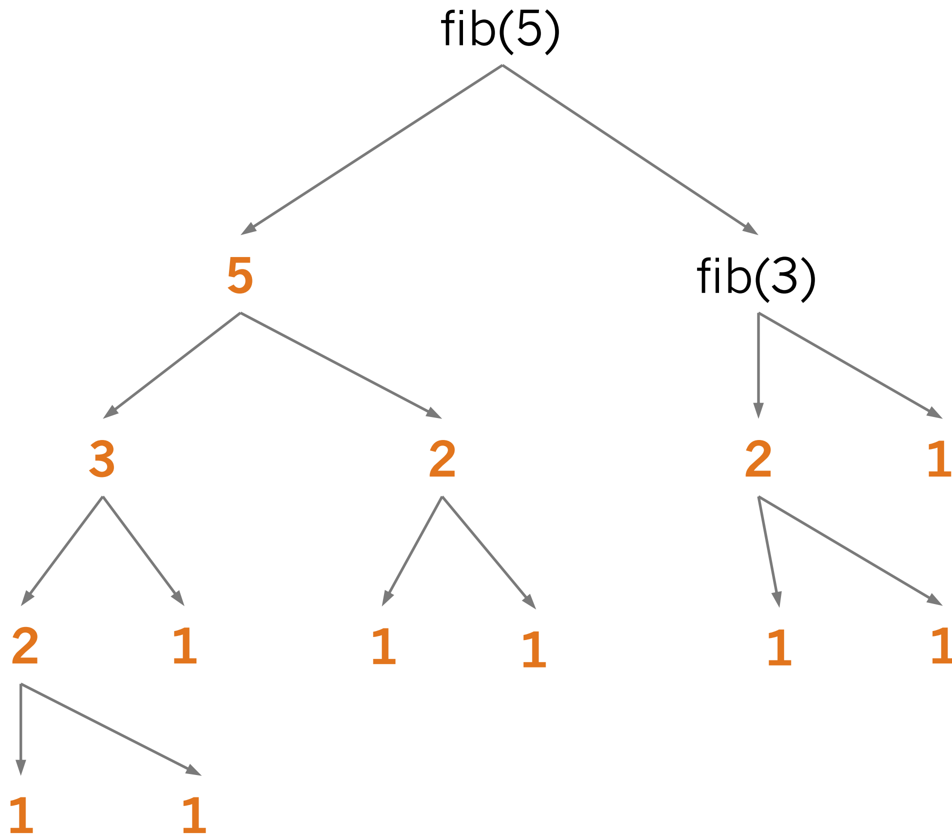
Line A → out1 = fib(n-1)
Line B → out2 = fib(n-2)
Line C → return out1 + out2



current line

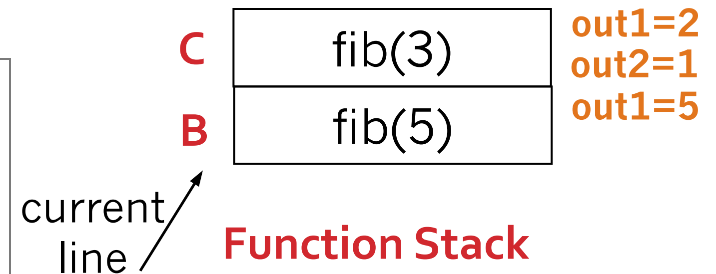
Function Stack

FIBONACCI FUNCTION STACK

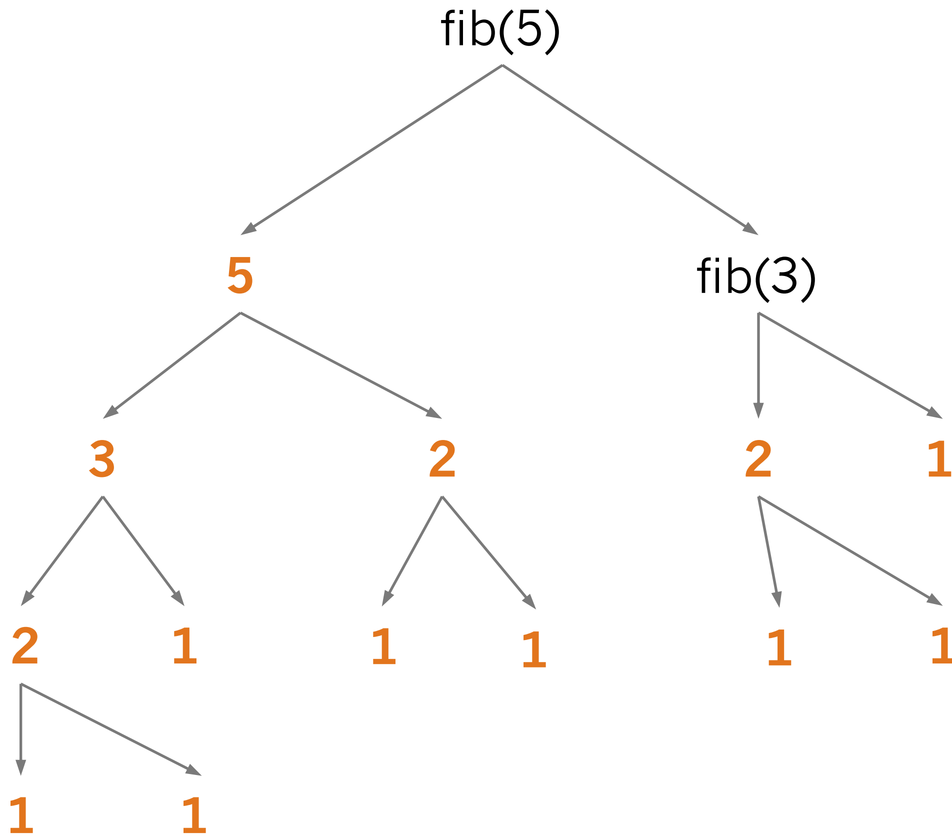


Line A → `out1 = fib(n-1)`
Line B → `out2 = fib(n-2)`
Line C → `return out1 + out2`

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

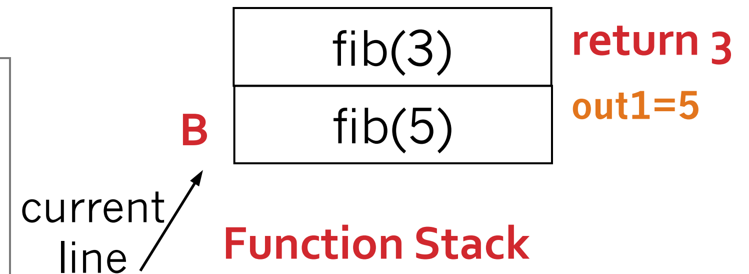


FIBONACCI FUNCTION STACK

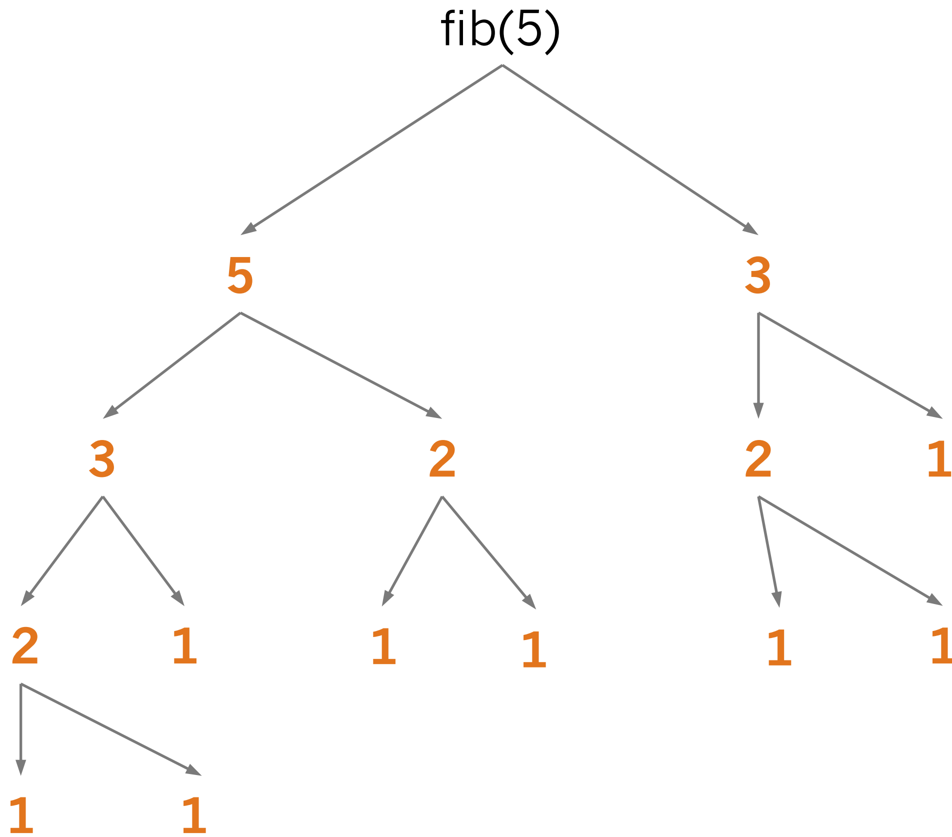


Line A → `out1 = fib(n-1)`
Line B → `out2 = fib(n-2)`
Line C → `return out1 + out2`

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

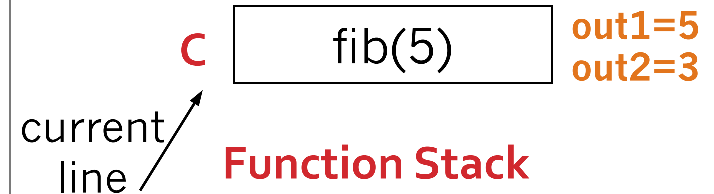


FIBONACCI FUNCTION STACK

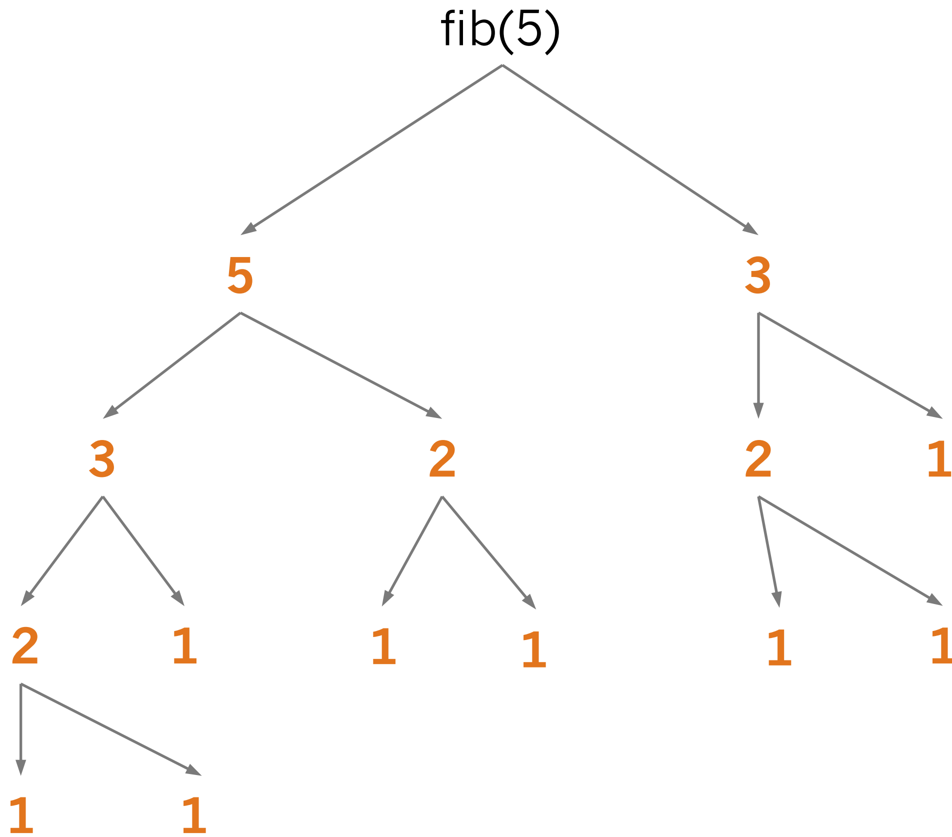



```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A → `out1 = fib(n-1)`
Line B → `out2 = fib(n-2)`
Line C → `return out1 + out2`



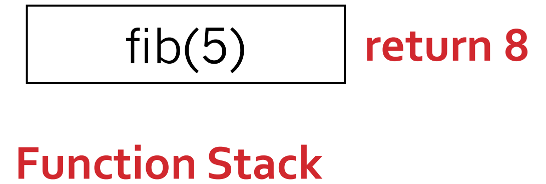
FIBONACCI FUNCTION STACK



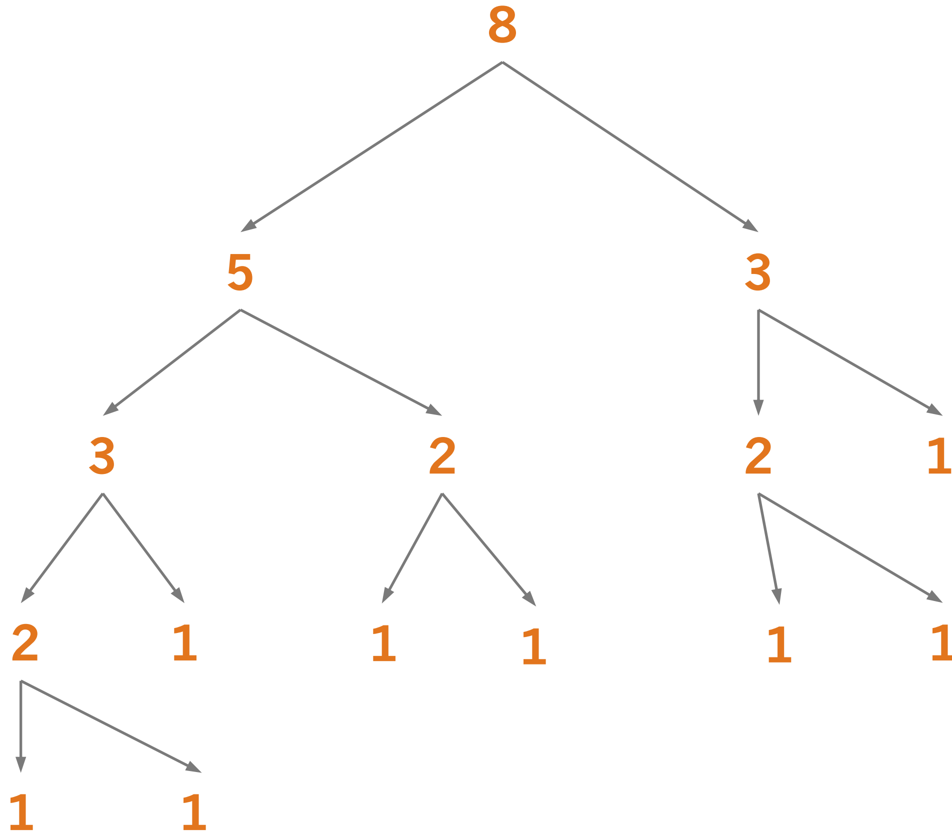
Line A
Line B
Line C

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

current line



FIBONACCI FUNCTION STACK



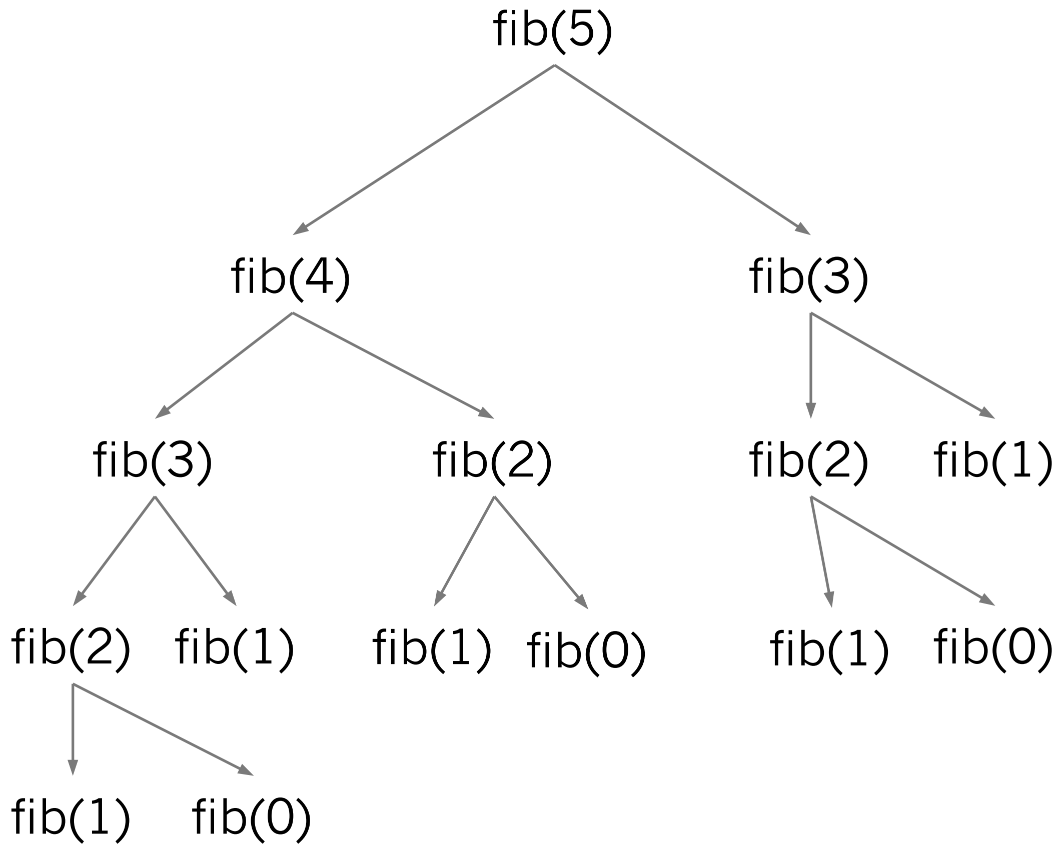

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        out1 = fib(n-1)
        out2 = fib(n-2)
        return out1 + out2
```

Line A → `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 → out1 = fib(n-1)
Line B → out2 = fib(n-2)
Line C → return out1 + out2

FEB 27 OUTLINE

- Review check-in, recap Stacks
- Lab 3 suggestions
- Abstract Data Types (ADTs) and interfaces
- Implementing stacks
- Queues (theory and implementation)

LAB 3 SUGGESTIONS

Top Down Design – think about what you want first, then implement it!

```
LinkedList femaleNames = new LinkedList();
LinkedList maleNames = new LinkedList();

// myEntries is from the CSV reader, of type ArrayList<String[]>
for (String[] row : myEntries) {
    maleNames.insertAlpha(row[1]);   // male name at index 1
    femaleNames.insertAlpha(row[3]); // female name at index 3
}

String flag = "-f";
String query = "Mary";
if (flag.equals("-f")) {
    femaleNames.printResult(query);
} else {
    maleNames.printResult(query);
}
```

Note: there are different/better ways than above.

LAB 3 SUGGESTIONS

Workflow:

* always be in a state where something is working
* make a small change, test, repeat
* don’t write lots of code without testing it!

```
LinkedList myList = new LinkedList();  // not the built-in, your own LinkedList class

myList.insertAlpha("Lizzie"); // don't have to call it insertAlpha
myList.insertAlpha("Will");
myList.insertAlpha("Juvia");
myList.insertAlpha("Emile");
myList.insertAlpha("Steve");
myList.insertAlpha("Gareth");

System.out.println(myList); // make sure you Override toString so this works
```

This should print (alphabetical order):

```
Emile, Gareth, Juvia, Lizzie, Steve, Will
```

FEB 27 OUTLINE

- Review check-in, recap Stacks
- Lab 3 suggestions
- **Abstract Data Types (ADTs) and interfaces**
- Implementing stacks
- Queues (theory and implementation)

ABSTRACT DATA TYPES (ADT)

- Mathematical/theoretical model of a data structure
- Specifies what data is stored
- Specifies the methods that operate on the data
- Says “what” but not “how”
- Practically: use an interface to specify ADT

See textbook: chapter 2 intro

INTERFACES REVISITED

Interfaces are a way to separate the ideas / goals of a class or set of methods from the implementation:

- a collection of methods (with full signature, but no body)
- no instance variables (except for static final constants)
- method modifiers necessary - implicitly public but okay to include
- no constructors and can not be instantiated
- a class *implementing* an interface must implement all methods as specified

INTERFACE EXAMPLE

```
public interface Shape {
  public double area();
}
```

INTERFACE EXAMPLE

```
public interface Shape {
  public double area();
}
```

```
public class Circle implements Shape {
      private double radius;
      public double area(){
            return Math.PI*radius*radius;
      }
}
```

INTERFACE EXAMPLE

```
public interface Shape {
  public double area();
}
```

```
public class Circle implements Shape {
      private double radius;
      public double area(){
            return Math.PI*radius*radius;
      }
}
```

```
public class Square implements Shape {
      private double sideLength;
      public double area(){
            return sideLength*sideLength;
      }
}
```

FEB 27 OUTLINE

- Review check-in, recap Stacks
- Lab 3 suggestions
- Abstract Data Types (ADTs) and interfaces
- **Implementing stacks**
- Queues (theory and implementation)

THE STACK ADT

Insertion and deletions are Last In First Out – LIFO

Insert at the top

Remove from the top

Operations

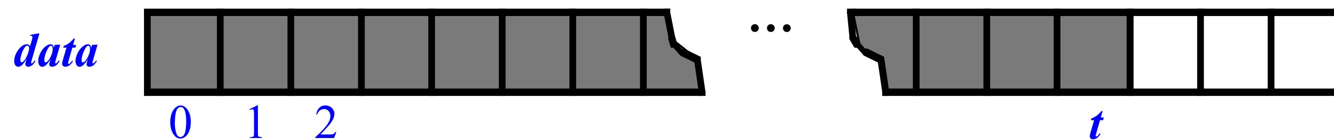
- `push(Object)`
- `Object pop()`
- `Object peek()`
- `int size()`
- `boolean isEmpty()`

ARRAY-BASED STACK

Implement the stack ADT with an array

Add elements onto the end of the array

Use an int `t` to keep track of the top



PERFORMANCE AND LIMITATIONS

Performance

- n: the number of objects in the stack
- Space used: $O(n)$
- Each operation runs in time $O(1)$

Limitations

- Max size is limited and can not be changed
- Pushing onto a full stack results in an implementation-specific exception

PERFORMANCE AND LIMITATIONS

Performance

- n: the number of objects in the stack
- Space used: $O(n)$
- Each operation runs in time $O(1)$

Limitations

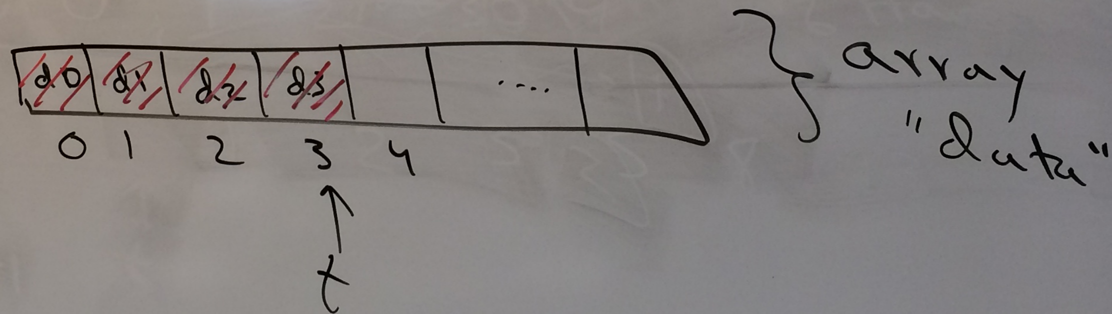
- Max size is limited and can not be changed
- Pushing onto a full stack results in an implementation-specific exception

Extra practice: how could you implement a Stack with a LinkedList?

① Make a new Java Project "StackQueuePractice"

② Make a new class ArrayStack

③ Make a new interface Stack



CODING EXERCISE

Make an ArrayStack class that implements a Stack interface:

```
public class ArrayStack<E> implements Stack<E>
```

- `push(E element)`
- `E pop()`
- `E peek()`
- `int size()`
- `boolean isEmpty()`

STACK INTERFACE FROM CLASS

```java
// abstract data type (ADT) Stack
public interface Stack<E> {

    // accessor methods
    public E peek();
    public int size();
    public boolean isEmpty();

    // modifier methods
    public E pop();
    public void push(E element);
}
```

STACK WITH ARRAY CODE

```
public class ArrayStack<E> implements Stack<E> {
  public static final int CAPACITY = 1000;
  private E[] data;
  private int t = -1;

  public ArrayStack(){this(CAPACITY);}

  public ArrayStack(int capacity) {
    data = (E[]) new Object[capacity]);
  }

  public int size() {return (t+1);}

  public boolean isEmpty() {return t == -1;}

  public E peek() {
    if (isEmpty()) {return null;}
    return data[t];
  }
```

STACK WITH ARRAY CODE

```
public E pop() {
  if isEmpty(){return null};
  E target = data[t];
  data[t] = null;     // garbage collection
  t--;
  return target;
}

public void push(E e) throws IllegalStateException {
  if (t == data.length-1) {
    throw new IllegalStateException("Stack is full");
  }
  else {
    // increment t then use t as index
    t += 1
    data[t] = e;
  }
}
```

FEB 27 OUTLINE

- Review check-in, recap Stacks
- Lab 3 suggestions
- Abstract Data Types (ADTs) and interfaces
- Implementing stacks
- **Queues (theory and implementation)**

QUEUES

How would you want a data structure to work for waiting in line at a store?

What is the rate of input is different than the rate of output?

Example: people show up to the DMV at random times, but processing takes about the same time for each person

Define an abstract data type (ADT).

THE QUEUE ADT

Insertions and deletions are First In First Out (FIFO)

Insert at the back

Delete from the front

Operations:

- `enqueue(Object)`
- `Object dequeue()`
- `Object first()`
- `int size()`
- `boolean isEmpty()`

IMPLEMENTING A QUEUE

Brainstorm: using the data structures we know about, how could we implement this ADT?

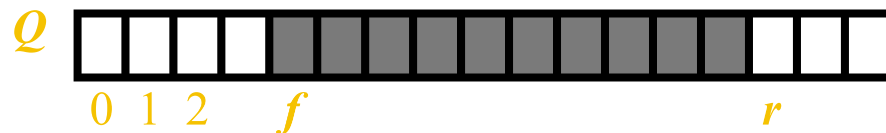
ARRAY-BASED QUEUE IMPLEMENTATION

An array of size `n` in a circular fashion

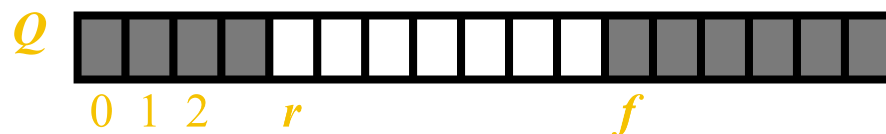
Two `int`s to track front and size

- `f`: index of the front element
- `size`: number of stored elements

normal configuration



wrapped-around configuration



DESIGNING DATA STRUCTURES

1. **Make a Course object that can store a name and list of students. Include relevant constructors, getters, and setters.**
2. **Make a LimitedEnrollmentCourse that has a cap on the number of students who can enroll. Have it inherit from Course.**
3. **Make addStudent, removeStudent, and getEnrolled methods that correctly handle limited versus unlimited enrollment.**

Extra practice!