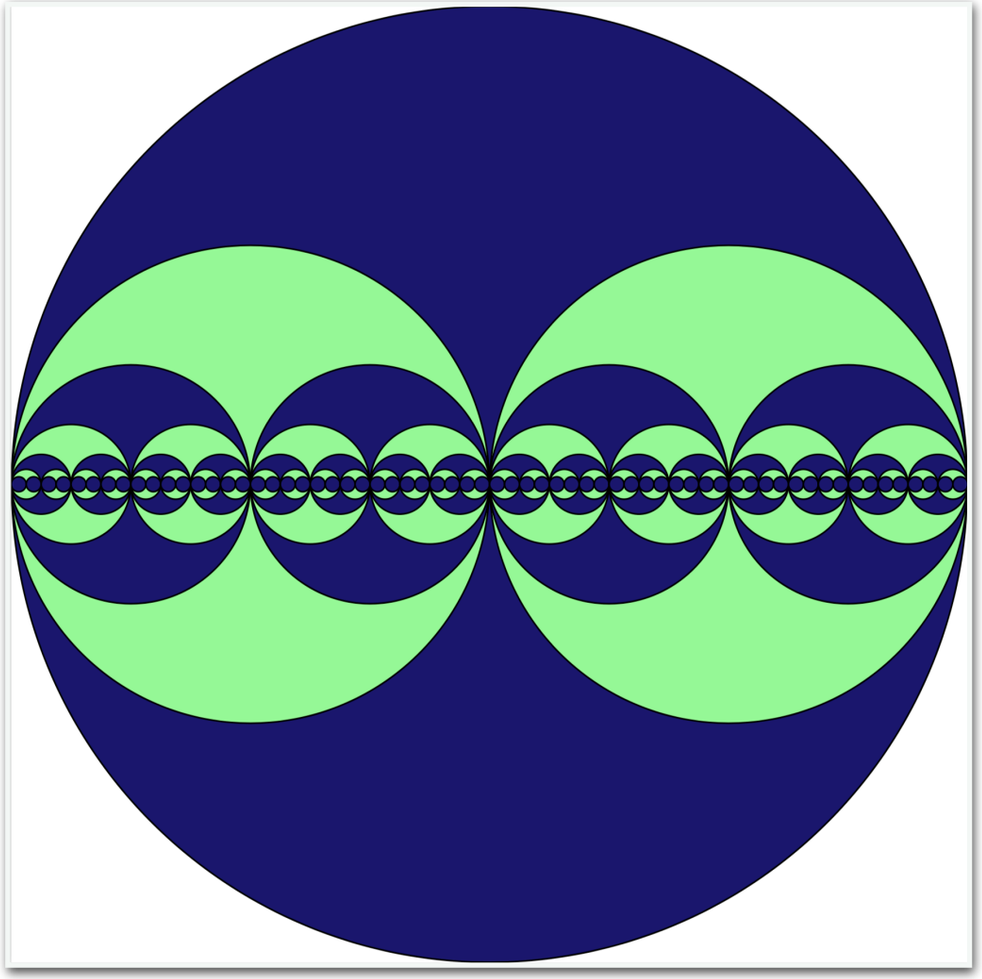
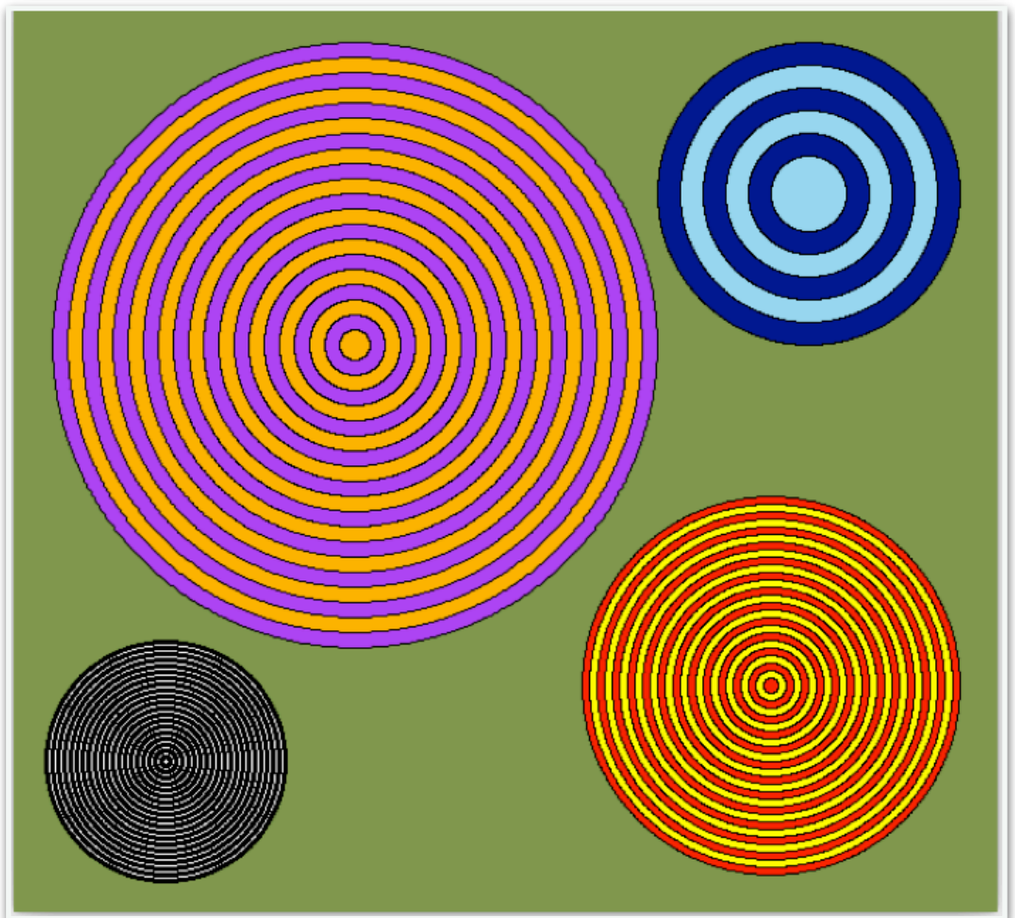
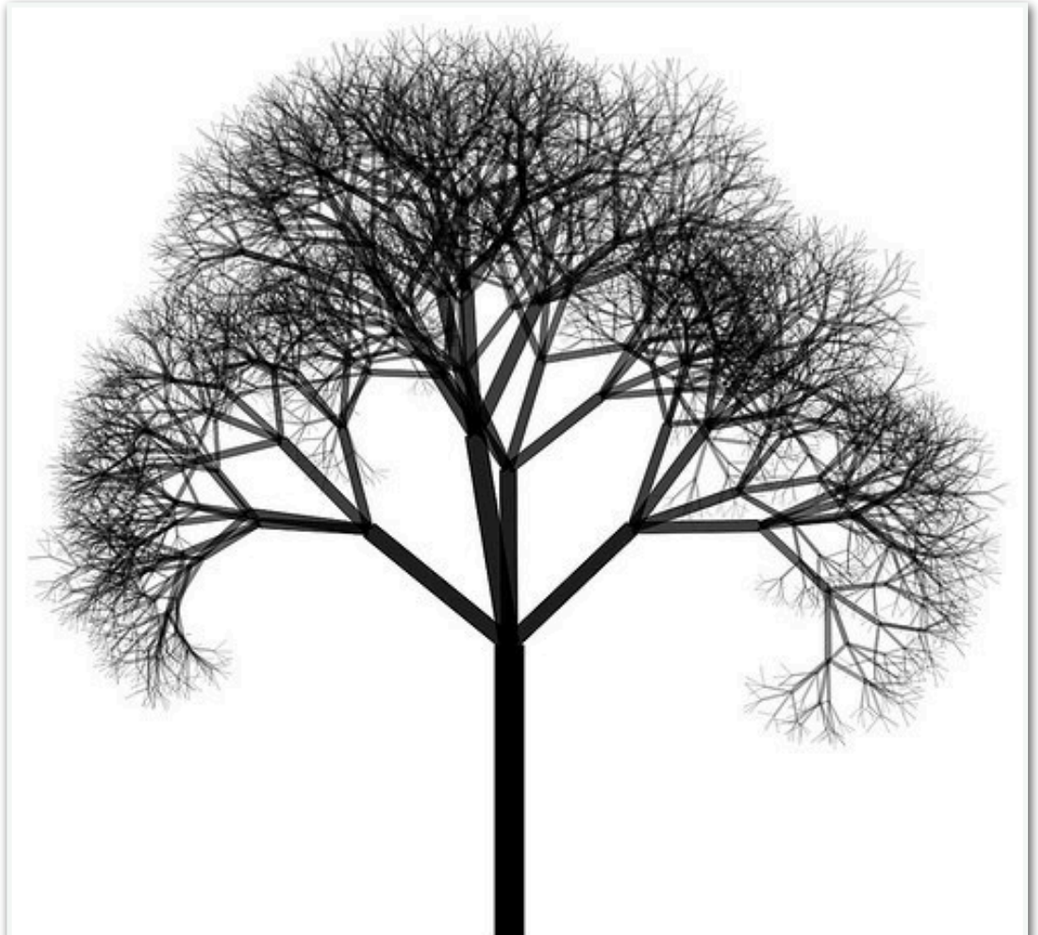
Introduction to Recursion







Recursion

- Recursion is the name given to a process that is composed of (smaller versions) of the itself

Recursive Approach

- REDUCE the problem to smaller subproblem(s) (smaller version(s) of itself)
- DELEGATE the smaller problems to the recursion fairy *(formally known as induction hypothesis)* and assume they're solved correctly
- COMBINE the solution(s) of the smaller subproblems to reach/return the solution

Palindromes

EVE

CIVIC

MADAM

AVID DIVA

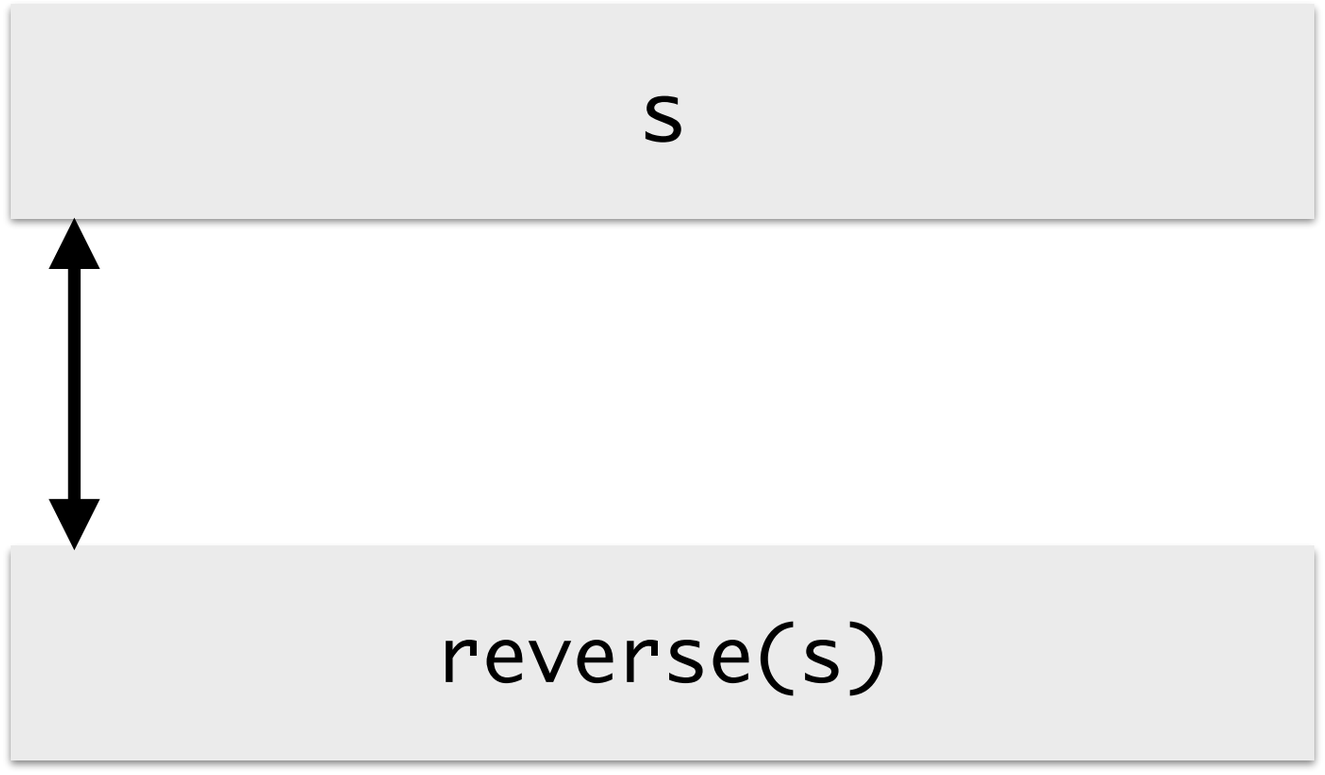
STEP ON NO PETS

STRESSED DESSERTS

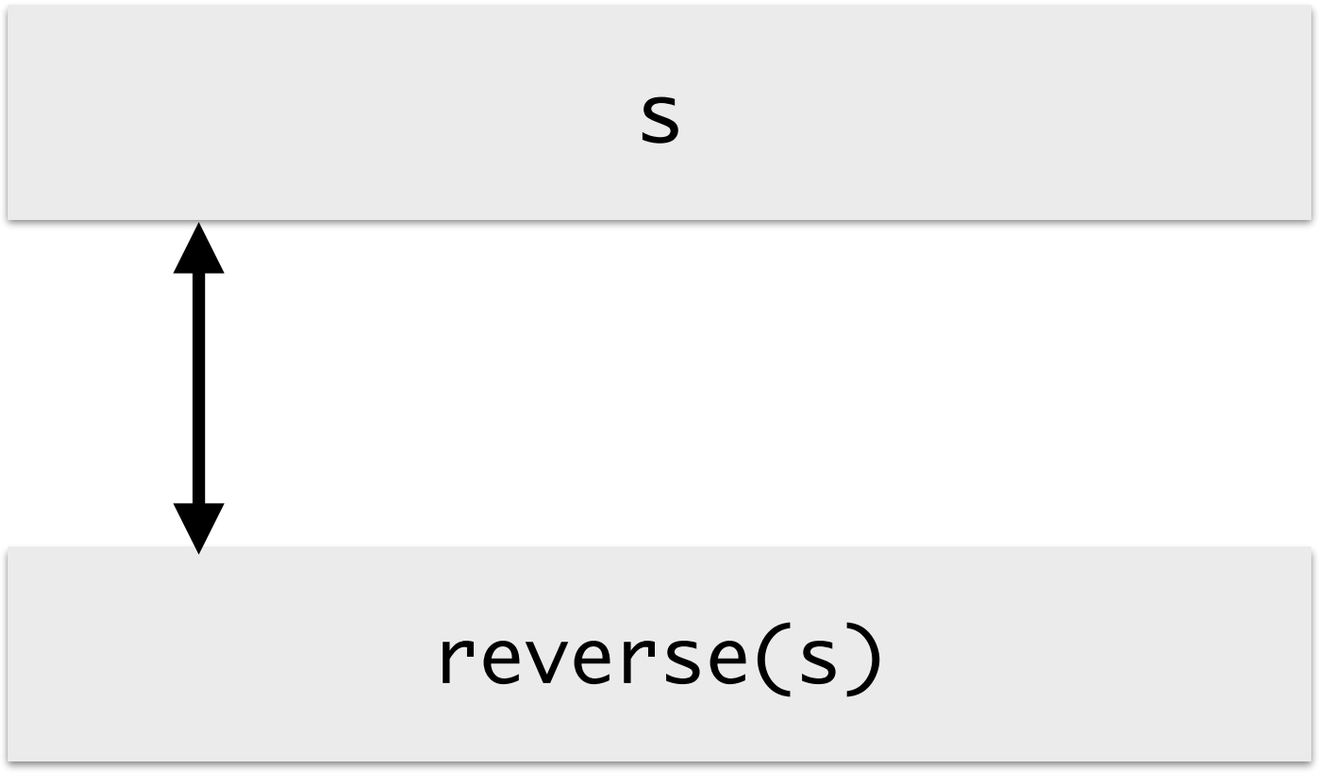
ABLE WAS I ERE I SAW ELBA

LIVED ON DECAF FACED NO DEVIL

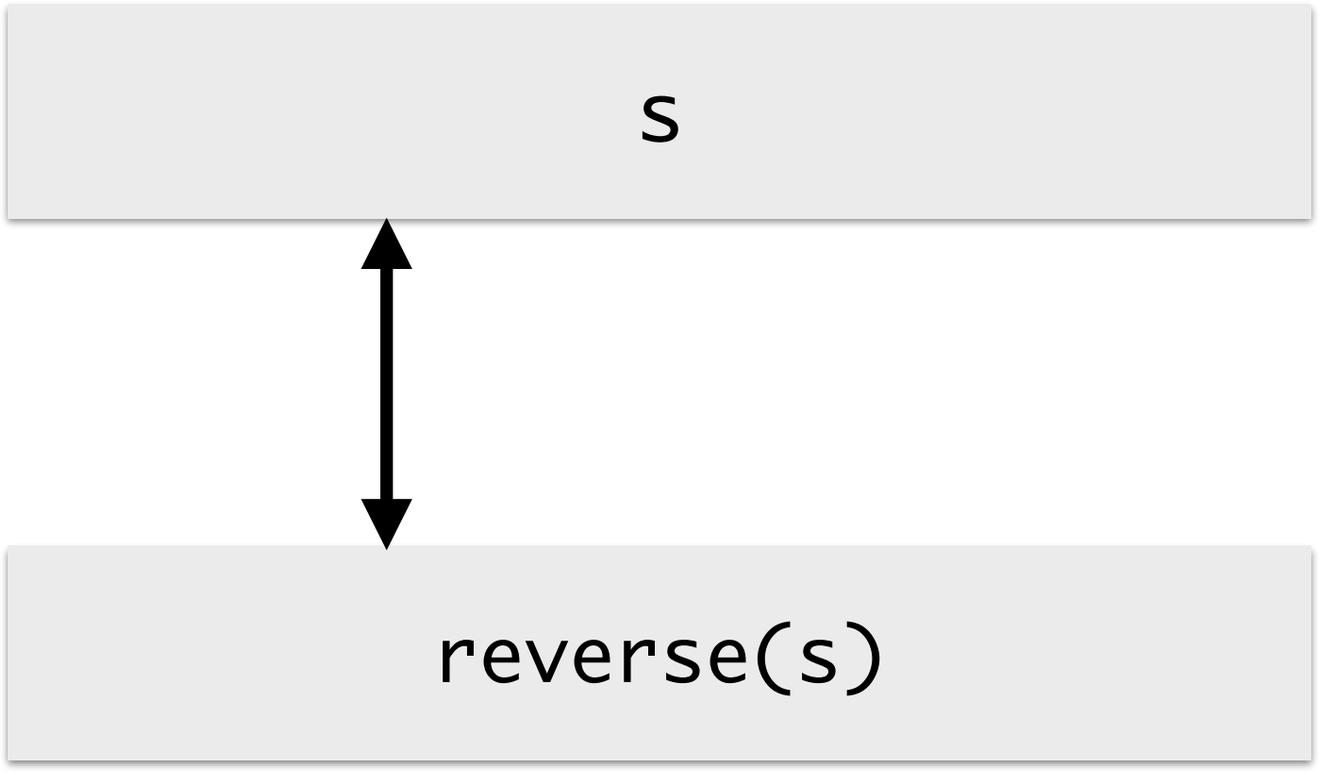
Iterative Approach



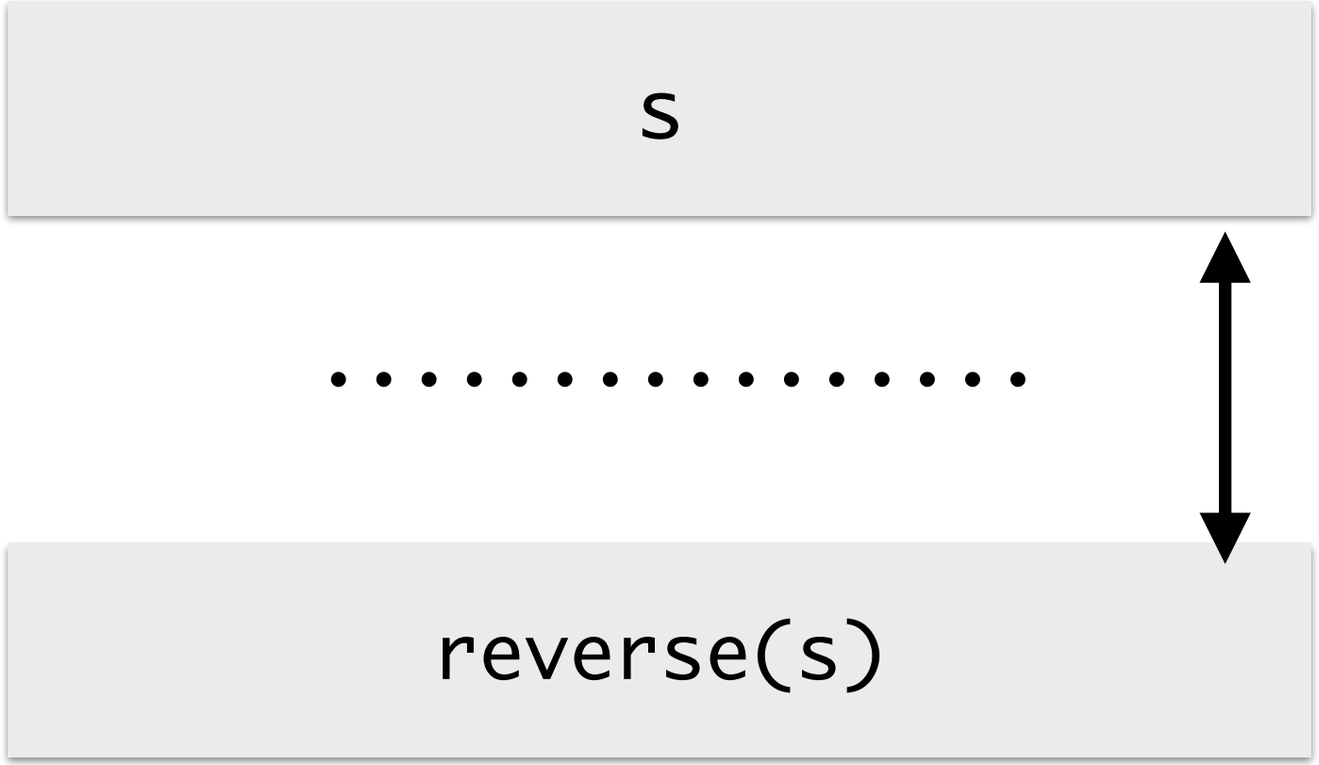
Iterative Approach



Iterative Approach

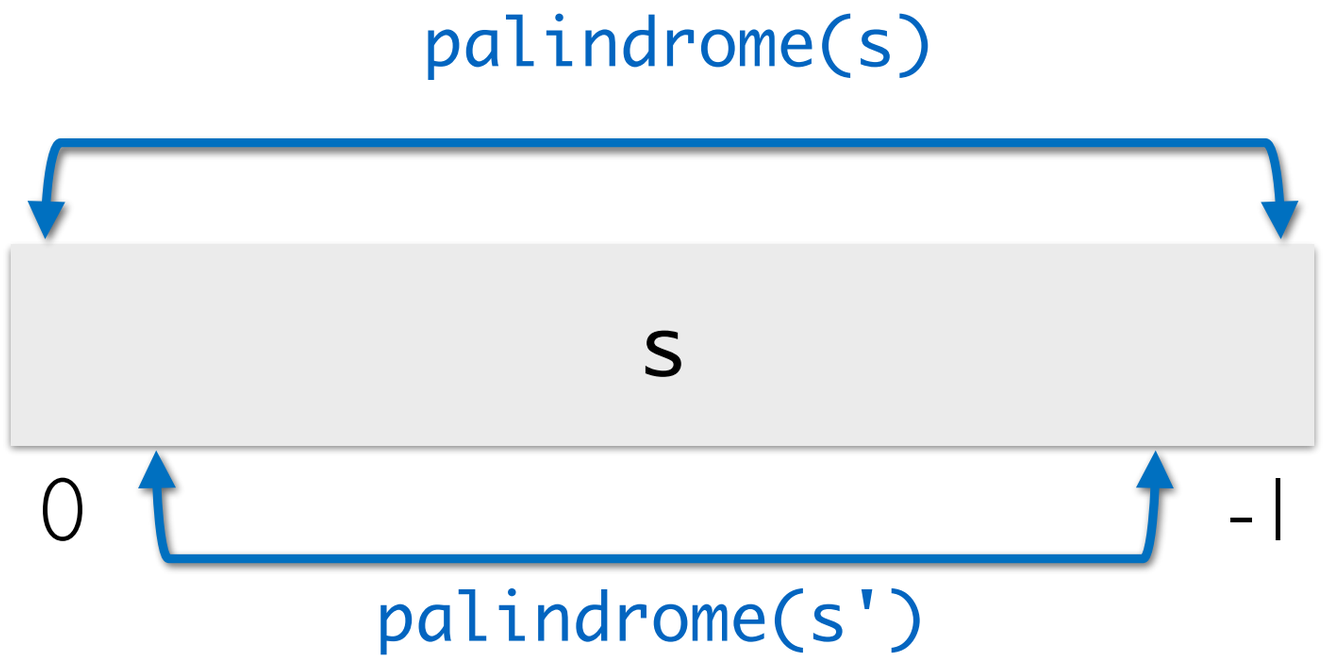


Iterative Approach



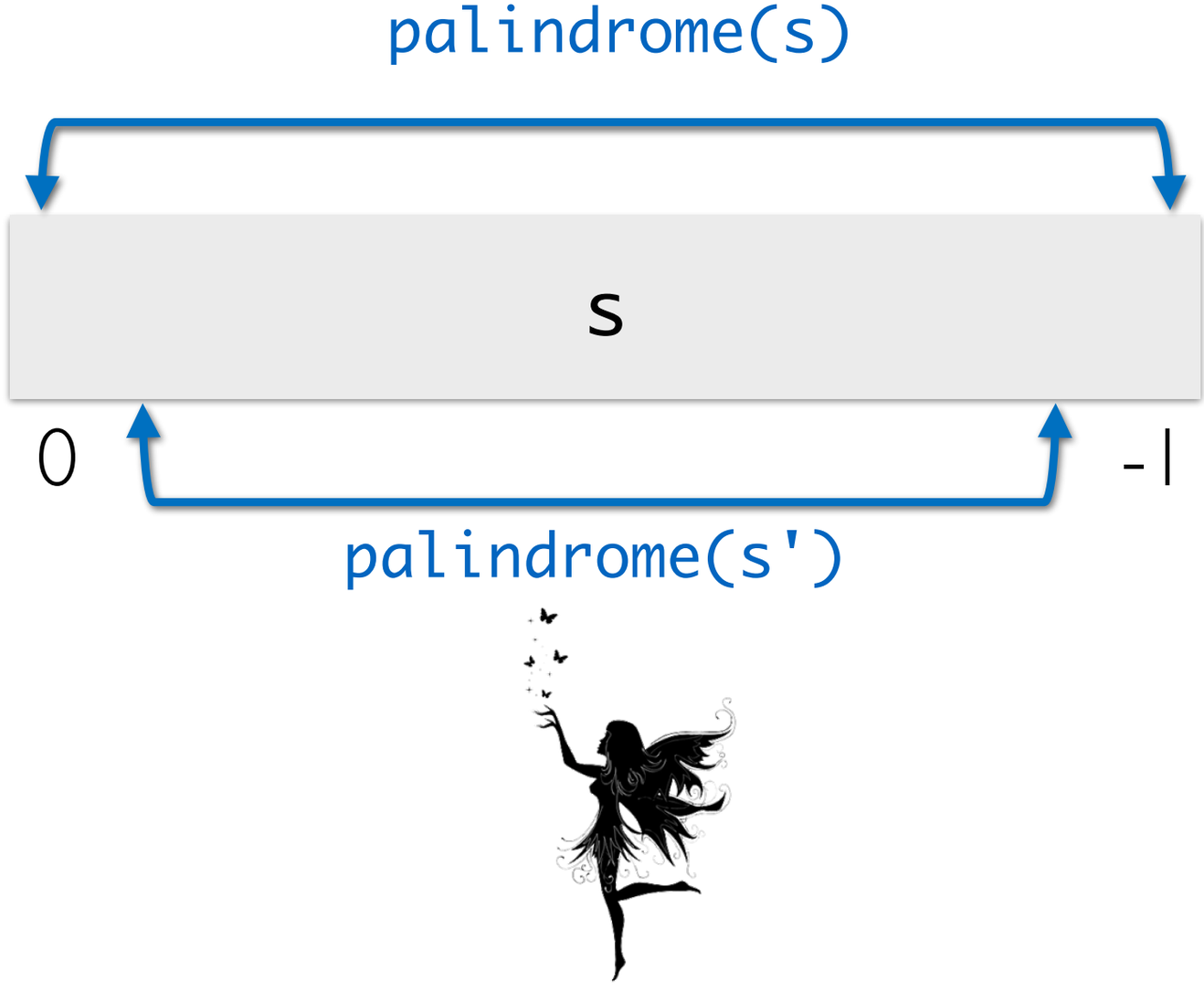
Recursive Approach

- REDUCE it smaller version of the same problem
  - Check if `s' = s[1:-1]` is a palindrome



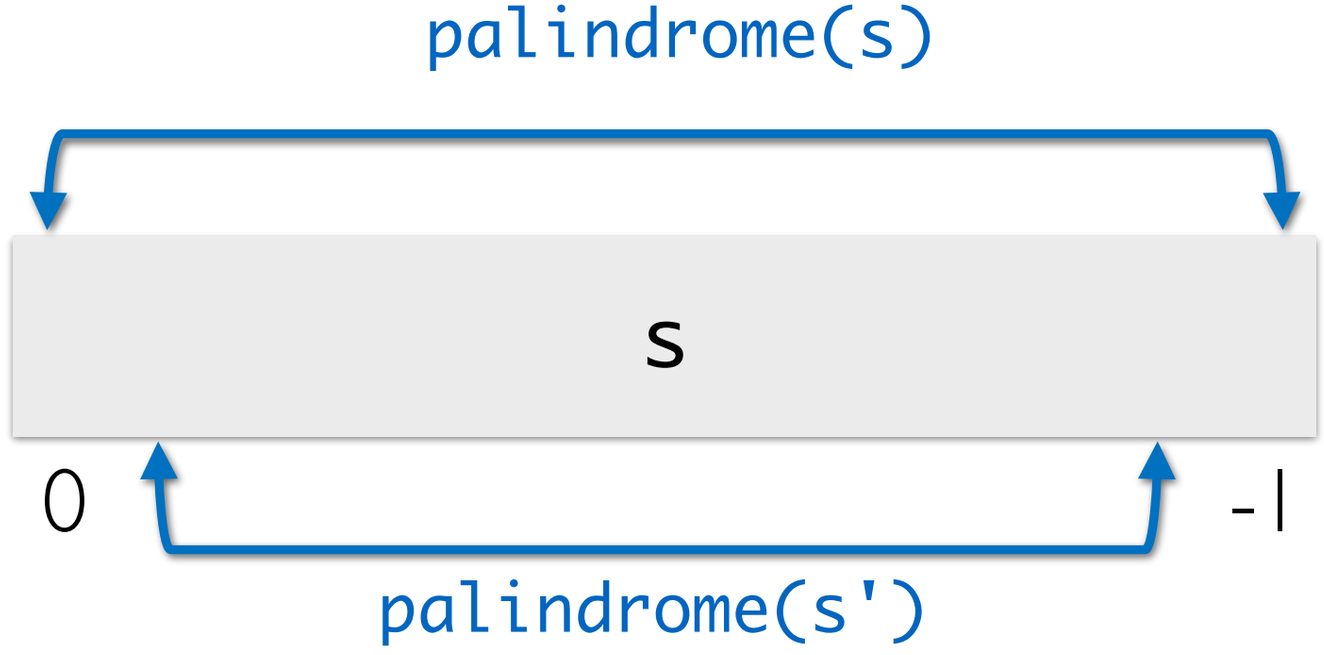
Recursive Approach

- DELEGATE the smaller problems to the recursion fairy ***(formally known as induction hypothesis)*** and assume they're solved correctly



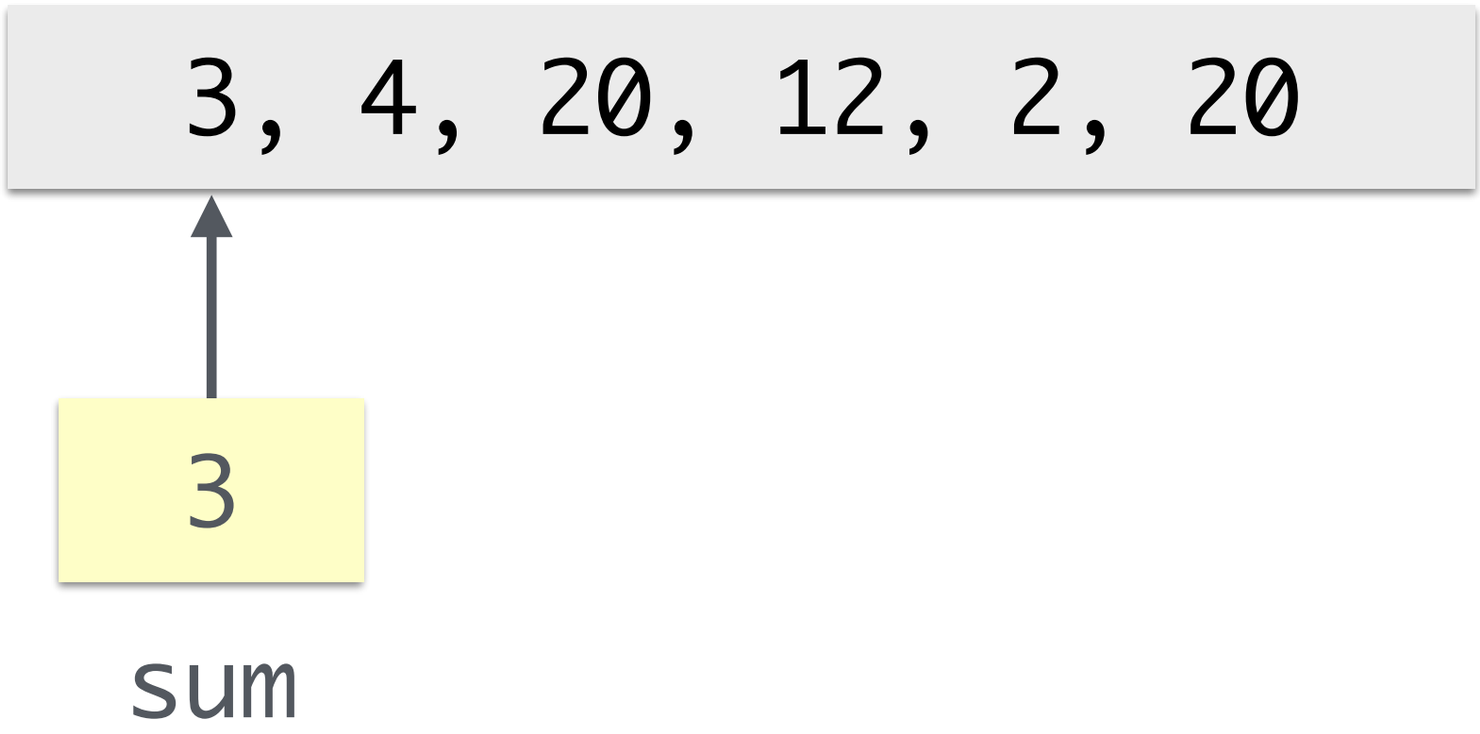
Recursive Approach

- COMBINE the solution(s) of the smaller subproblems to reach/return the solution
  - return `True` if `palindrome(s')` is `True` and `s[0]` is same as `s[-1]`

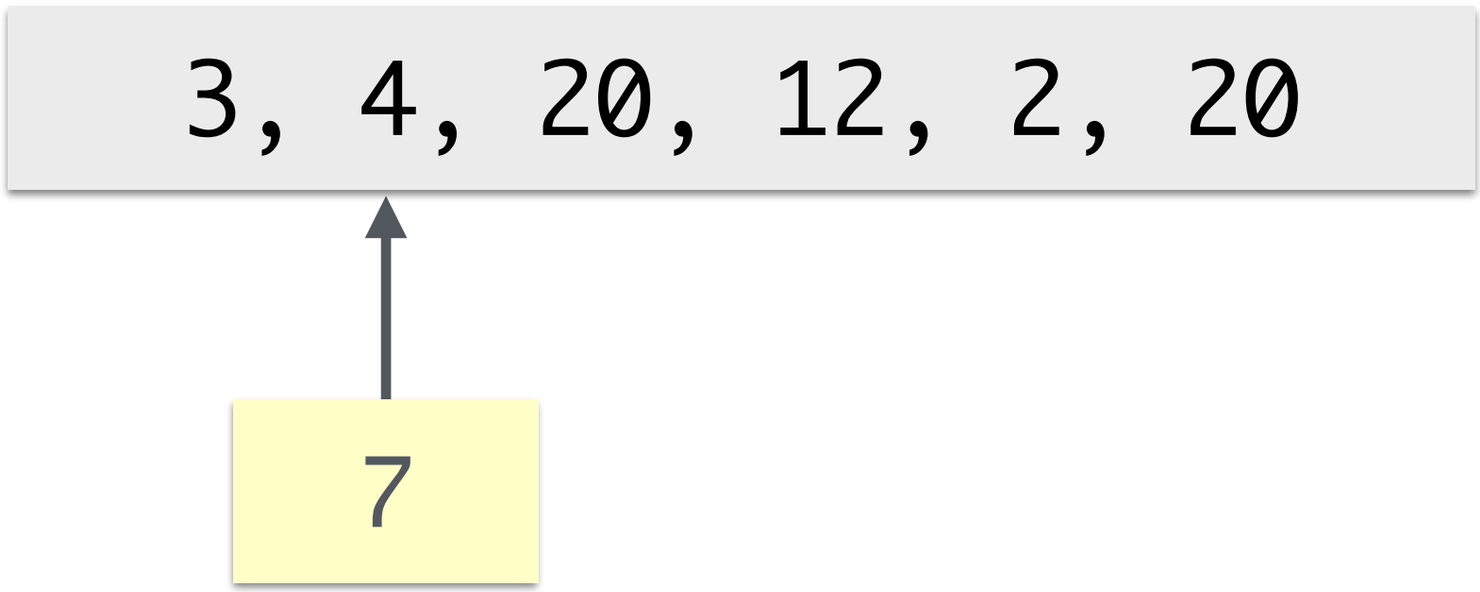


Summing Numbers in a List

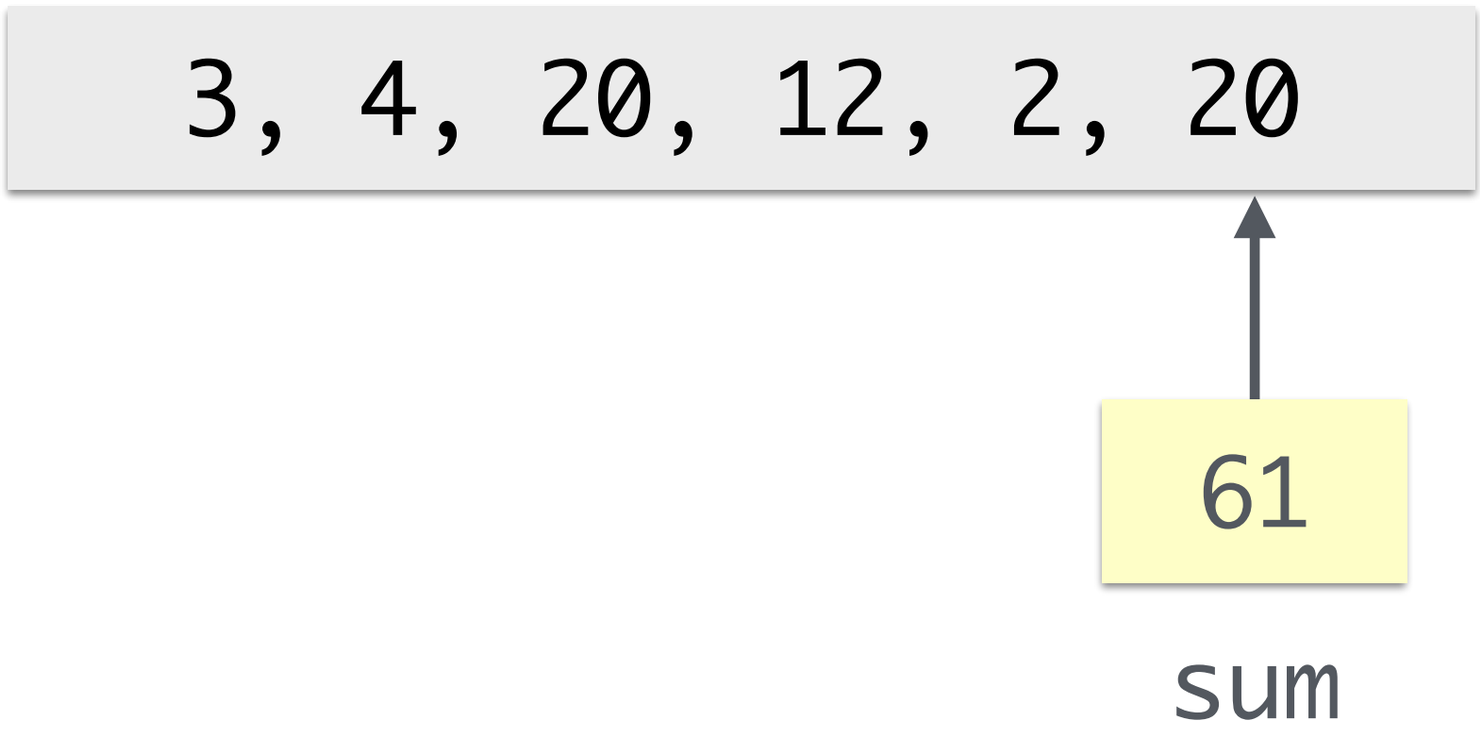
Iterative Approach



Iterative Approach

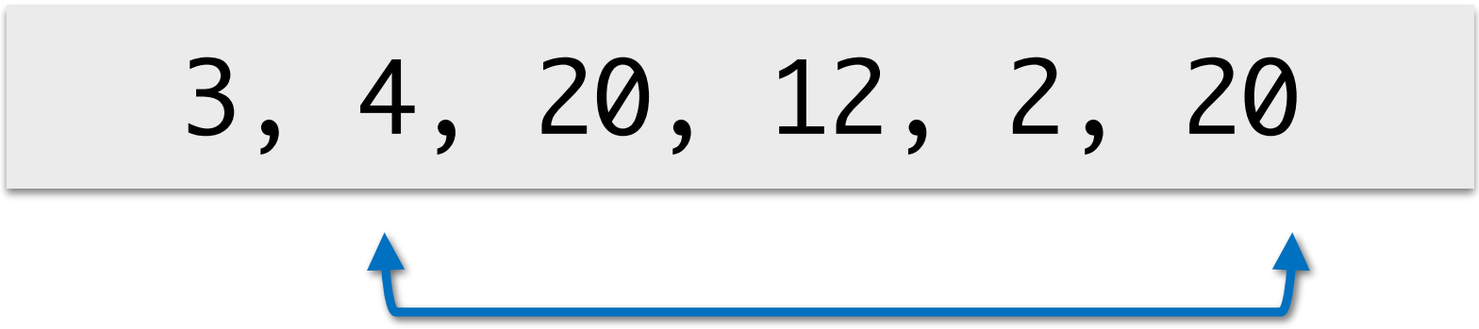


Iterative Approach



Recursive Approach

- REDUCE it smaller version of the same problem
  - sum `numList[1:]`



Recursive Approach

- DELEGATE the smaller problems to the recursion fairy ***(formally known as induction hypothesis)*** and assume they're solved correctly

```
3, 4, 20, 12, 2, 20
```

```
sum(numList[1:])
```

Recursive Approach

- COMBINE the solution(s) of the smaller subproblems to reach/return the solution
  - return `numList[0] + sum(numList[1:]`

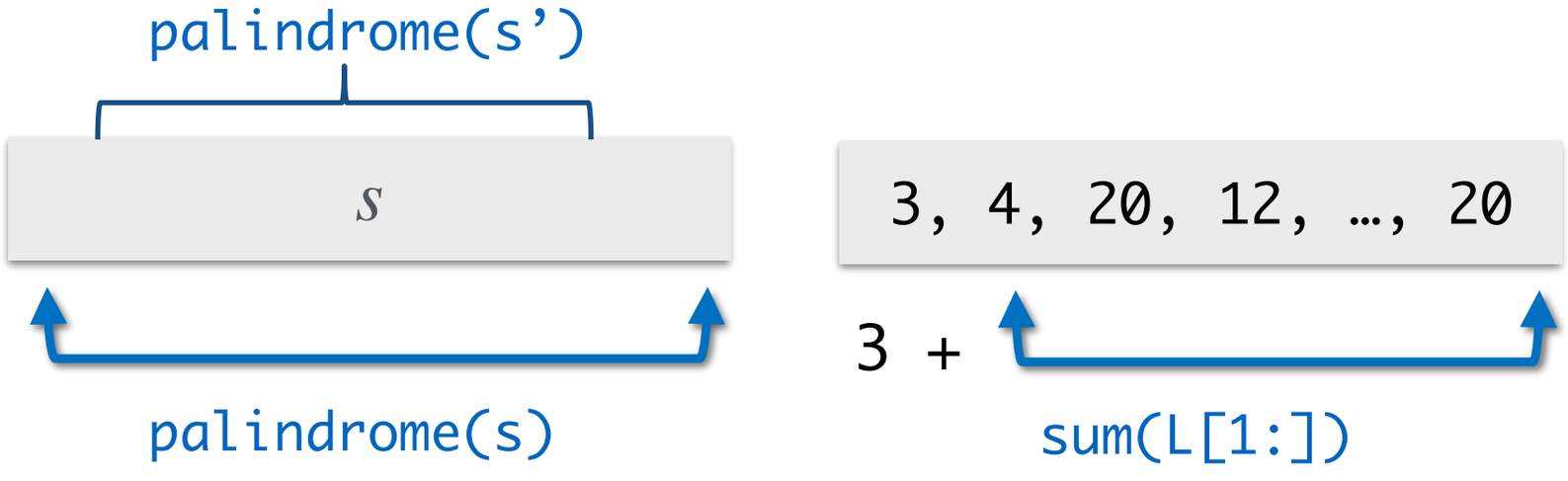
```
3, 4, 20, 12, 2, 20
```

3 +

`sum(numList[1:])`

Base Case(s):
the buck stops here

- Reached when the problem is sufficiently small that its trivial to solve directly
- Palindrome: empty string and all strings of length 1 are trivially palindromes
- Summing a list of numbers: the sum is trivially zero when the list is empty



Recursive Algorithm

- Base case: Solving problem directly.
- Recursive case:
  - REDUCE the problem to smaller subproblem(s) (smaller version(s) of itself)
  - DELEGATE the smaller problems to the recursion fairy *(formally known as induction hypothesis)* and assume they're solved correctly
  - COMBINE the solution(s) of the smaller subproblems to reach/return the solution

Recursive Functions

A recursive function is a function that calls itself

countDown

- Let's write a recursive function that prints integers from `n` down to `1` (**without using any loops**)

- Recursive case. (REDUCE/ DELEGATE/ COMBINE):
  Can think of `countDown(5)`
  as `print(5)` followed by `countDown(4)`

```
In[1] countDown(5)

 5
 4
 3
 2
 1
```

```
In[2] countDown(4)

 4
 3
 2
 1
```

countDown: Base Case

- **Base case.** When the problem is so simple that we can solve it without decomposing any further

```
In[1] countDown(0)
```

Nothing is printed

```
def countDown(n):
    '''Prints ints from n down to 1'''
    if n < 1:
        pass # do nothing
```

`countDown`: Recursive Case

- Recursive step does two things:
  - performs an action that contributes to the solution
  - Invokes the function on a smaller subproblem

```
def countDown(n):
    '''Prints ints from n down to 1'''
    if n < 1:
        pass # do nothing
    else:
        print(n)
        countDown(n-1)
```

`countDown`: Recursive Case

- Recursive step does two things:
  - performs an action that contributes to the solution
  - Invokes the function on a smaller subproblem

```
def countDownImplicitBaseCase(n):
    '''Prints ints from n down to 1'''
    if n > 0:
        print(n)
        countDown(n-1)
```

Structure of Recursion

- All recursive functions must have two types of cases
- **Base case.** A case where problem is so simple, its solution can be returned directly
- **Recursive case.** A case where the problem is
  - Decomposed into smaller subproblems, where at least of the subproblems is solved by invoking the function it
  - Trust the *recursion fairy* (aka *mathematical induction*) that smaller subproblems are solved correctly

Recursion GOTCHAs!

GOTCHA #1

- **Problem** that you are solving recursively is **not getting smaller**, that is, not getting closer to the base case --- *infinite recursion*!
- **Does not reach the base case**

```
def countDownGotcha(n):
    '''Prints ints from n down to 1'''
    if n < 1:
        pass # do nothing
    else:
        print(n)
        countDownGotcha(n)
```

Subproblem not getting smaller!

GOTCHA #2

- Missing base case/ unreachable base case---another way to cause *infinite recursion*!

```
def printHalvesGotcha(n):
    if n > 0:
        print(n)
        printHalvesGotcha(n/2)
```

Always true!

"Maximum recursion depth exceeded"

- In practice, the infinite recursion examples will terminate when Python runs out of resources for creating function call frames, leads to a `"maximum recursion depth exceeded"` error message

Review: Function Frame Model

A model to understand what happens we call a function

Review:
Function Frame Model

- Consider a simple function `square`
- What happens when `square(5)` is invoked?

```
def square(x):
    return x*x
```

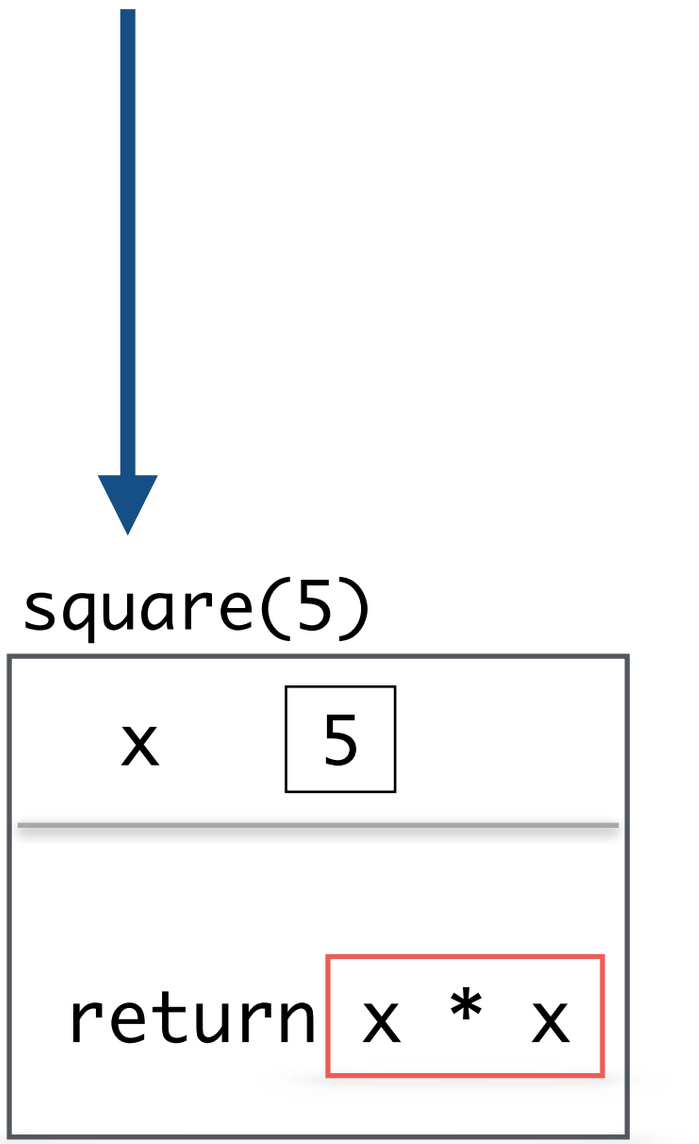
Review:
Function Frame Model

```
>>> square(5)
```

Review:

Function Frame Model

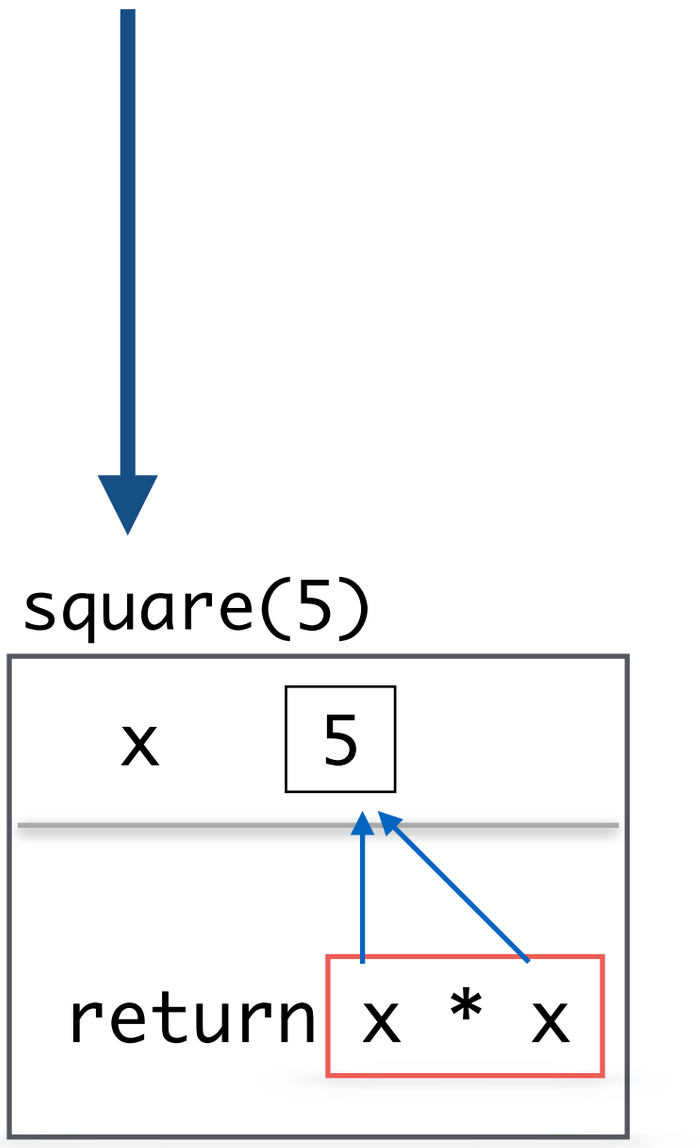
```
>>> square(5)
```



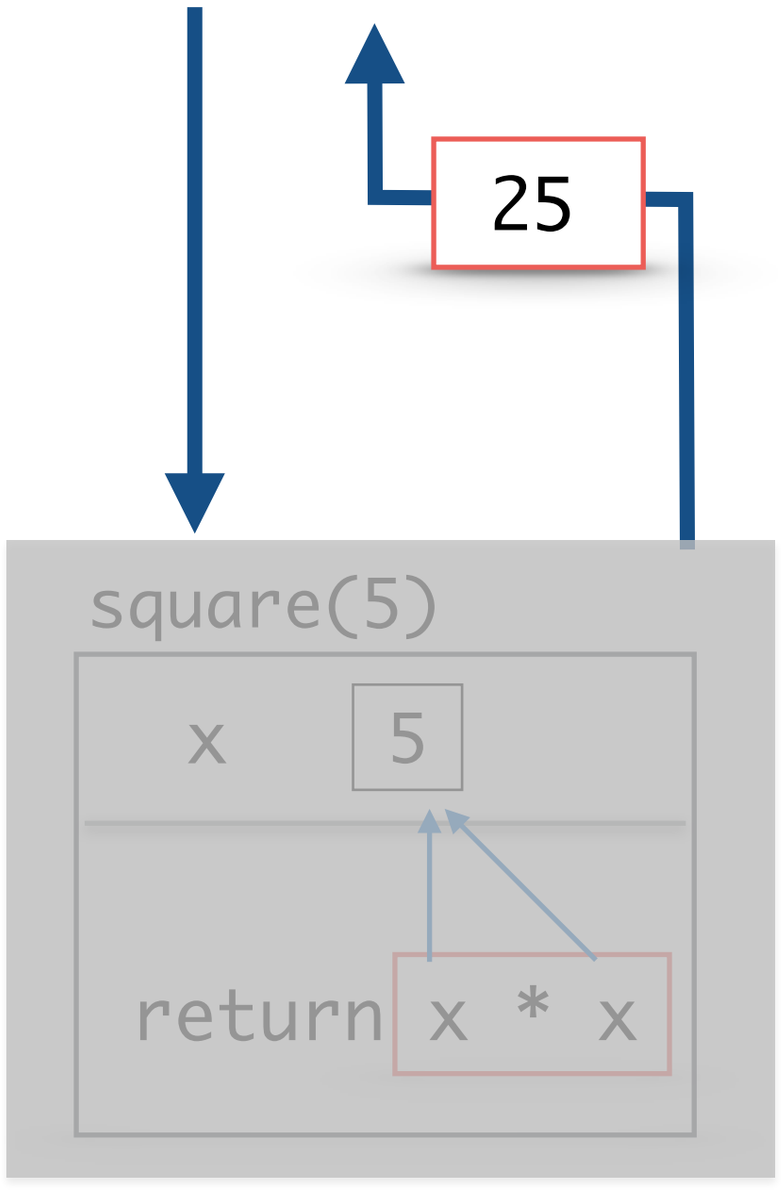
Review:

Function Frame Model

```
>>> square(5)
```

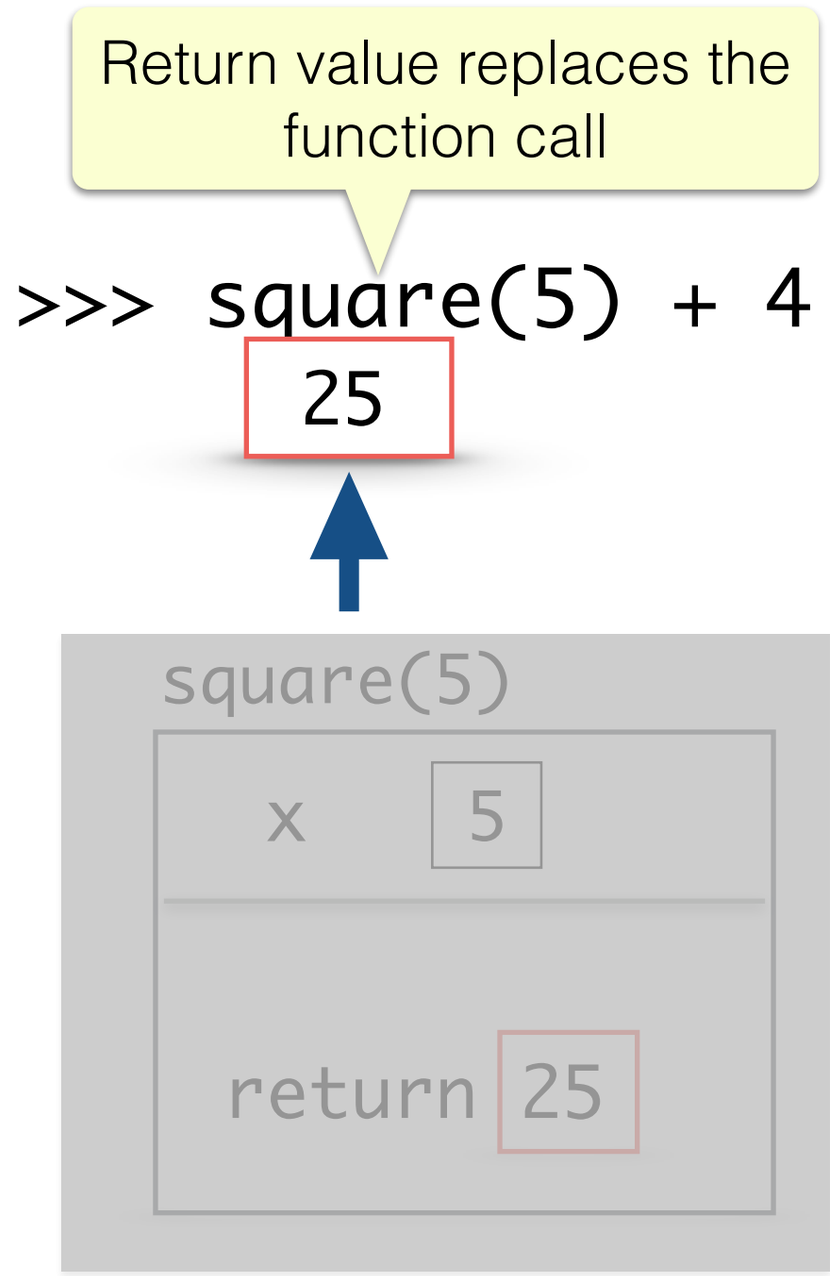


Review:
Function Frame Model



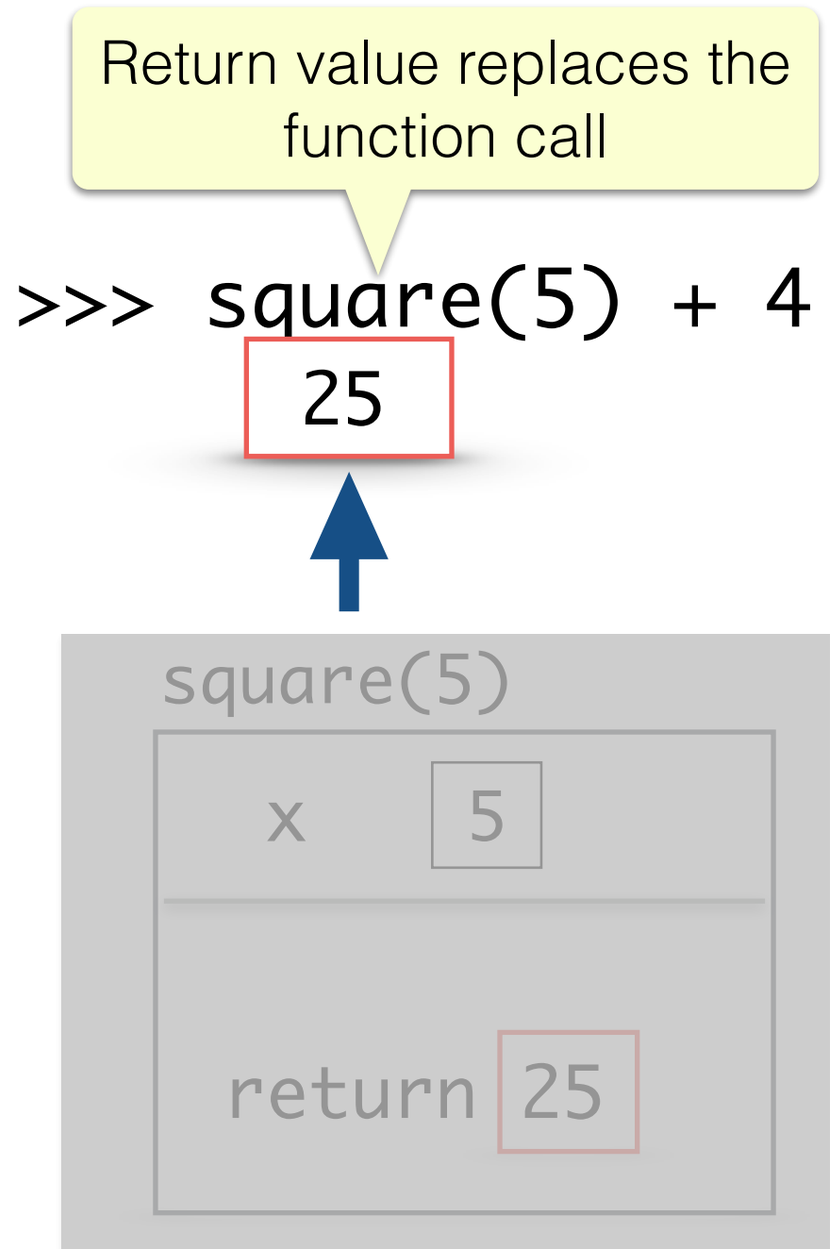
Summary:
Function Frame Model

- When we return from a function frame "control flow" goes back to where the function call was made
- The function frame (the state of the local variables inside it) **are destroyed after the return**



Summary:
Function Frame Model

- When we return from a function frame "control flow" goes back to where the function call was made
- The function frame (the state of the local variables inside it) **are destroyed after the return**
- If a function does not have an explicit return statement, then we return **None** when after all statements in the function body are executed



Review:

Function Frame Model

Review:
Function Frame Model

- How about functions that call other functions?

Review:
Function Frame Model

- How about functions that call other functions?

```
def sumSquare(a, b):

   return square(a) + square(b)
```

Review:
Function Frame Model

- How about functions that call other functions?

```
def sumSquare(a, b):

   return square(a) + square(b)
```

- What happens when we call `sumSquare(5, 3)`?

```
def sumSquare(a, b):

   return square(a) + square(b)
```

```
>>> sumSquare(5,3)
```

```
def sumSquare(a, b):

   return square(a) + square(b)
```

```
>>> sumSquare(5,3)
```

```
def sumSquare(a, b):

   return square(a) + square(b)
```

```
>>> sumSquare(5,3)
```

sumSquare(5, 3)

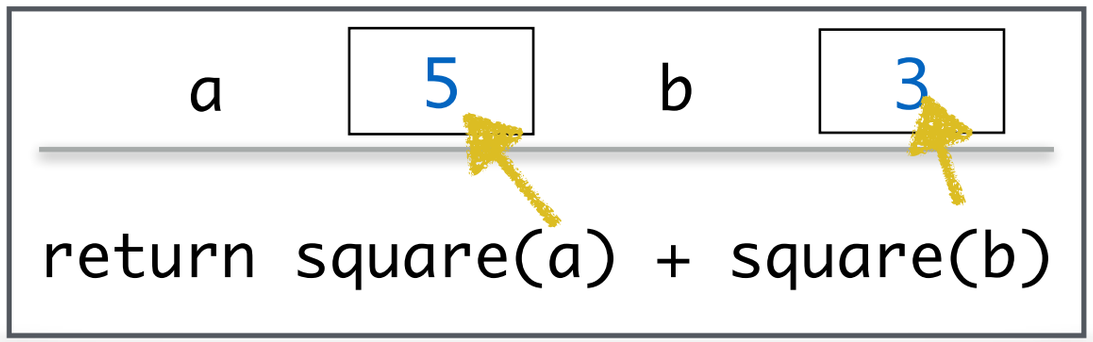
a	5	b	3

```
return square(a) + square(b)
```

```
def sumSquare(a, b):

   return square(a) + square(b)
```

```
>>> sumSquare(5,3)
```

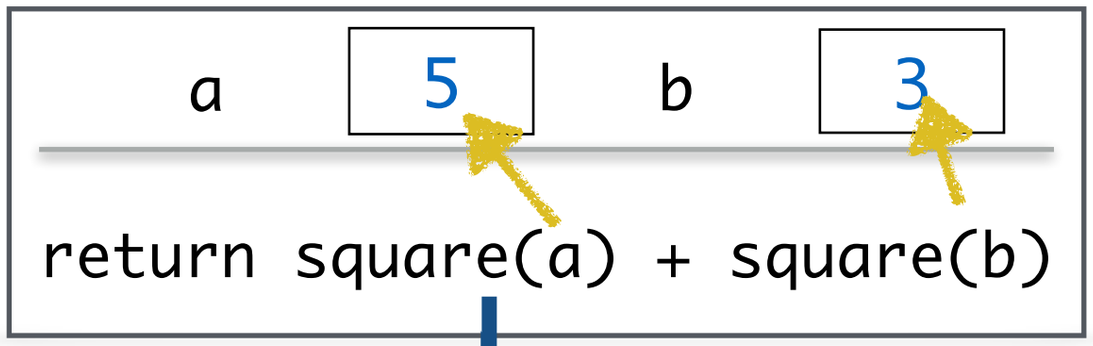
```
sumSquare(5, 3)
```



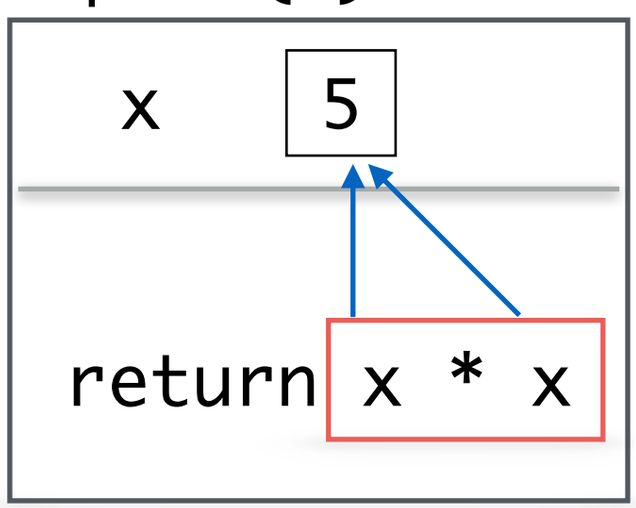
```
def sumSquare(a, b):
    return square(a) + square(b)
```

```
>>> sumSquare(5,3)
```

sumSquare(5, 3)

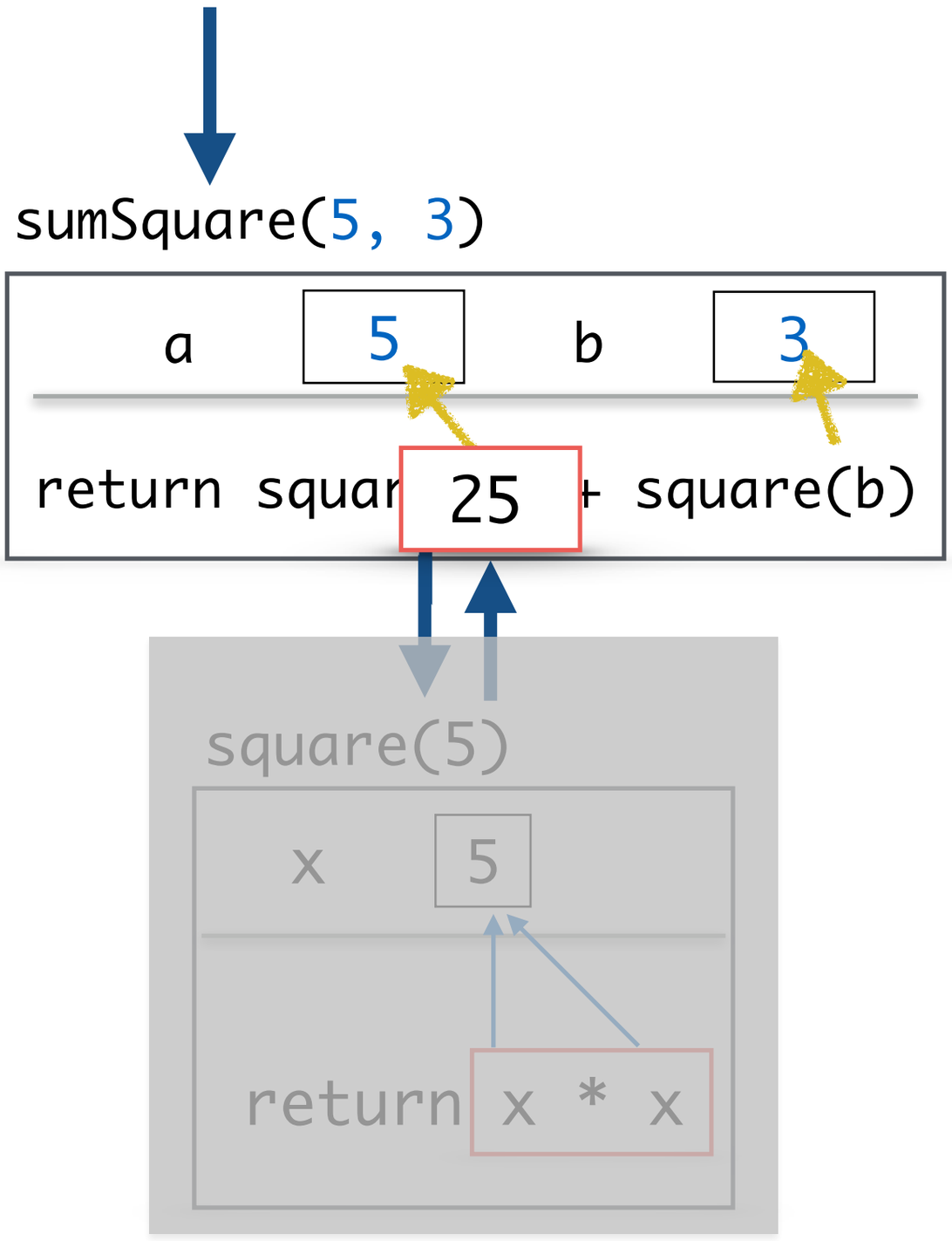


square(5)



```
def sumSquare(a, b):

   return square(a) + square(b)
```

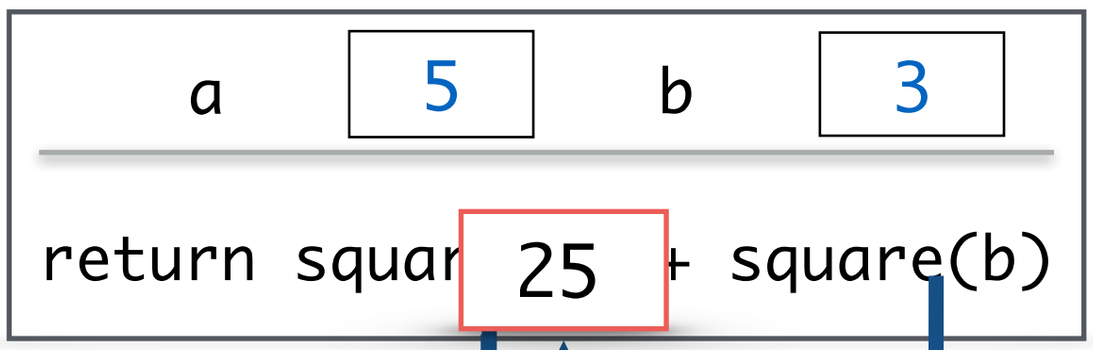
```
>>> sumSquare(5,3)
```

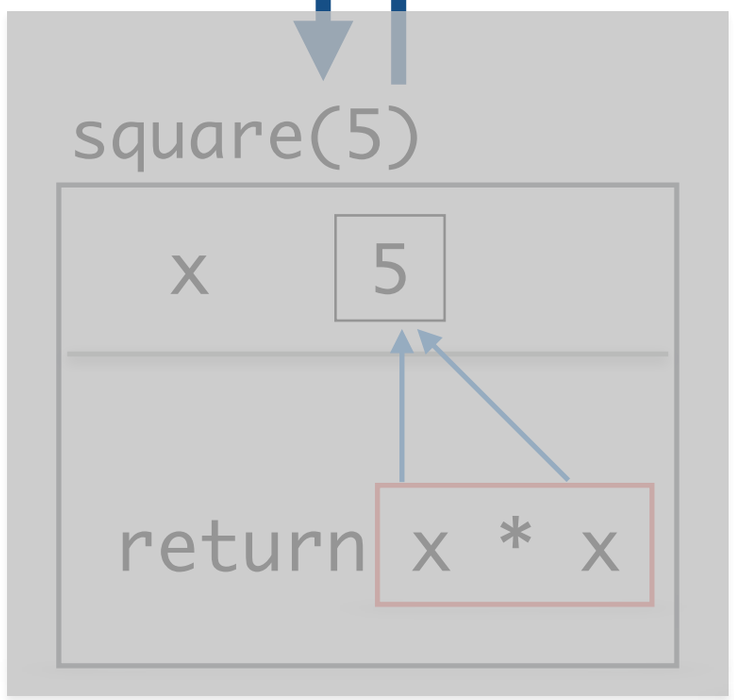


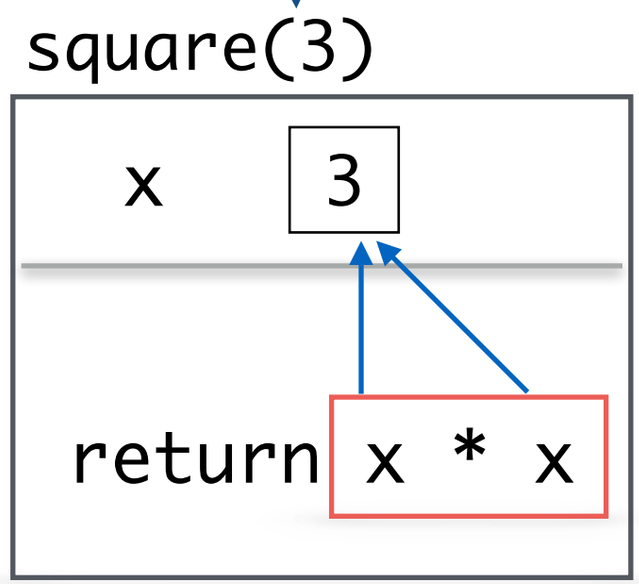
```
def sumSquare(a, b):
    return square(a) + square(b)
```

```
>>> sumSquare(5,3)
```

sumSquare(5, 3)







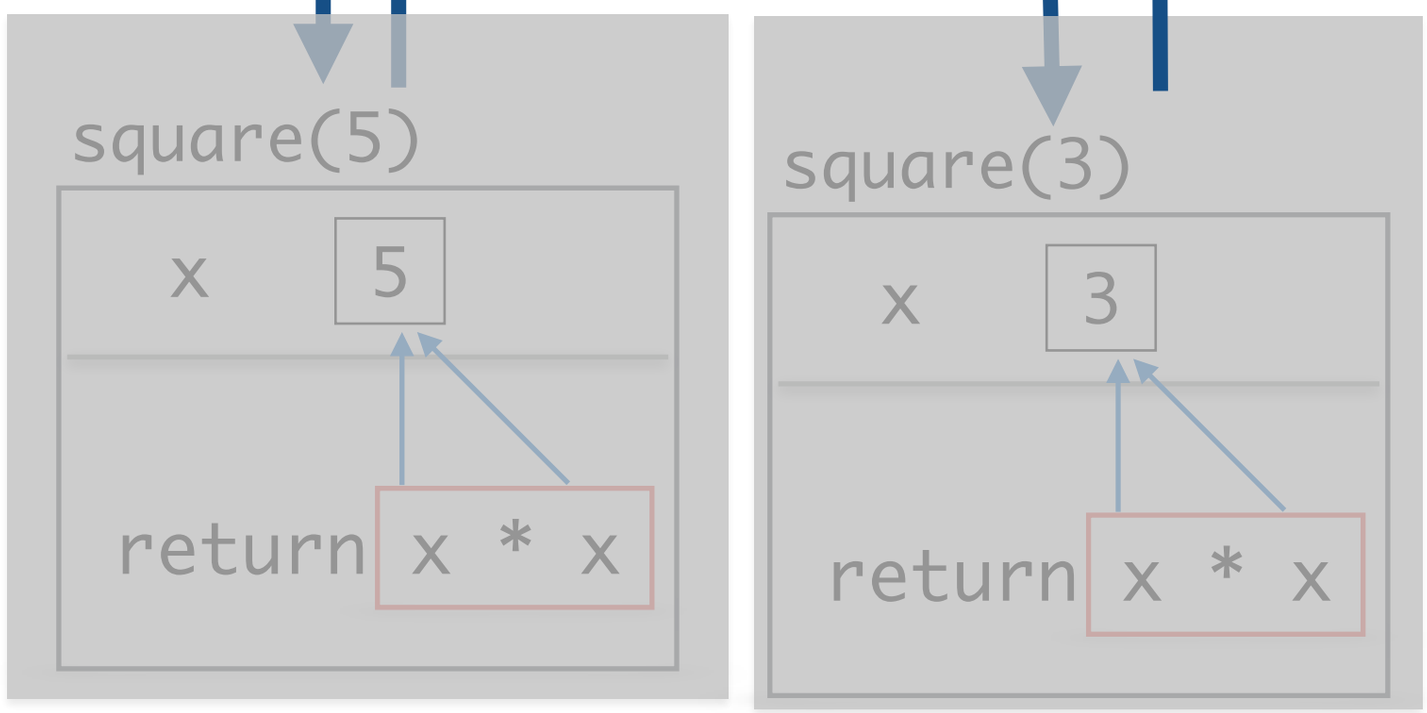
```
def sumSquare(a, b):
    return square(a) + square(b)
```

```
>>> sumSquare(5,3)
```

sumSquare(5, 3)

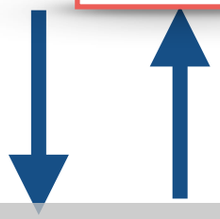
a	5	b	3

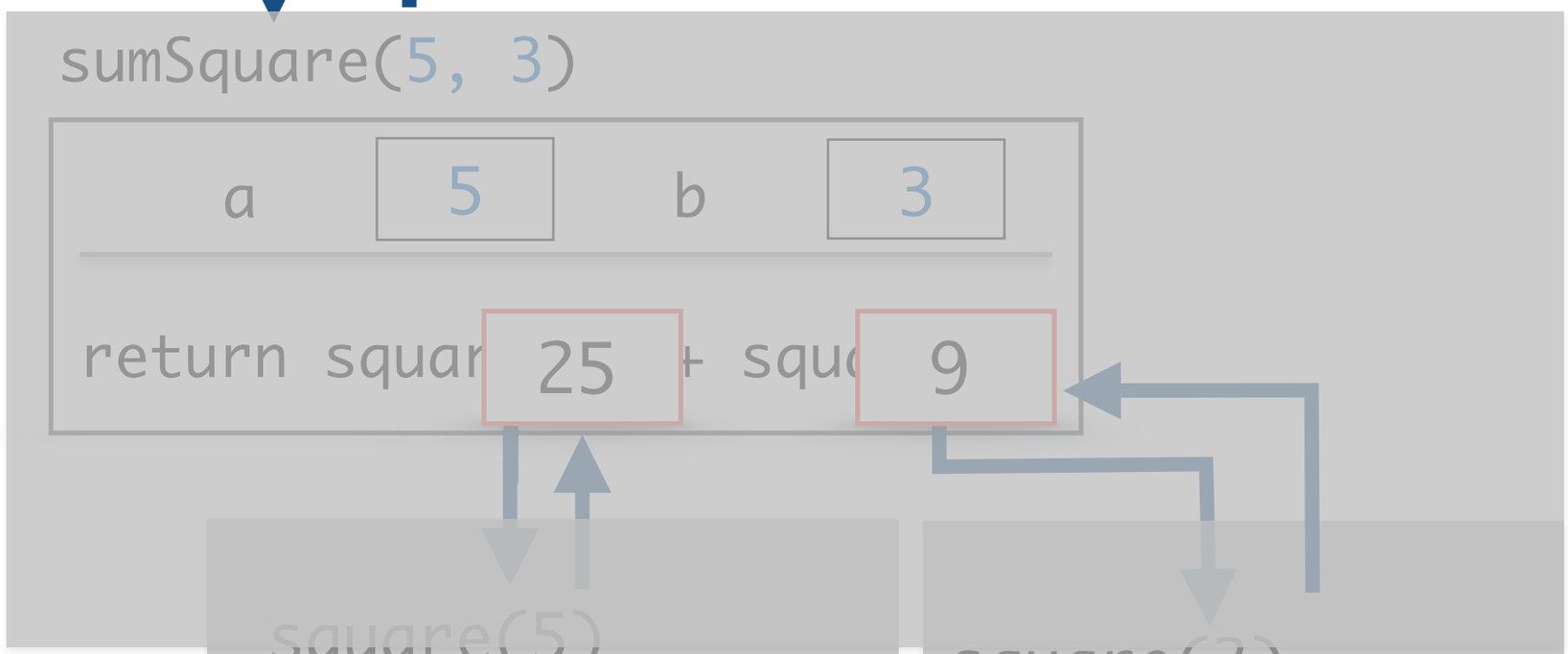
return squar 25 + squ 9

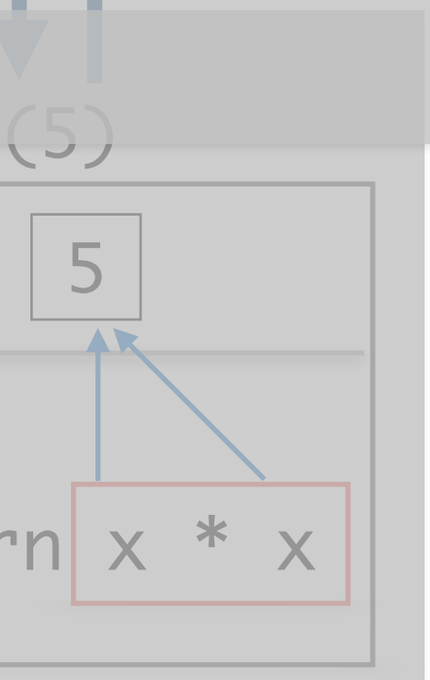


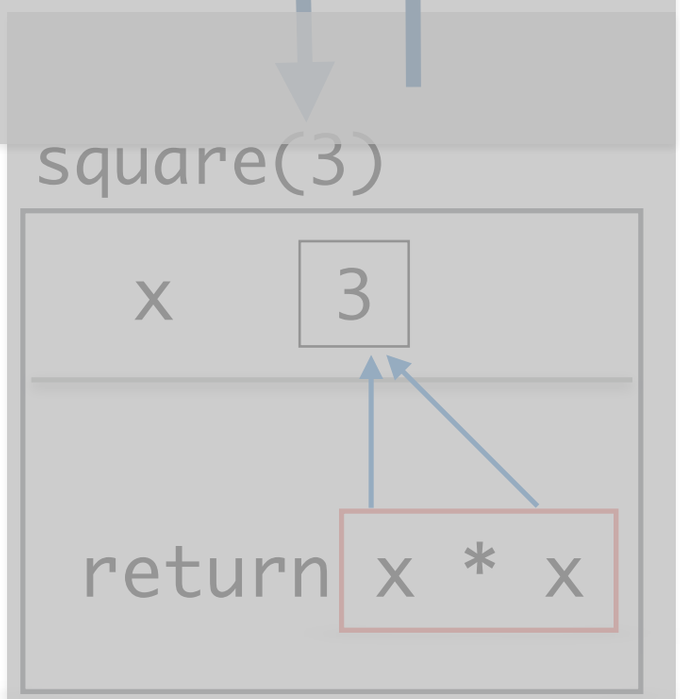
```
def sumSquare(a, b):

    return square(a) + square(b)
```

```
>>> sum 34 re(5,3)
```









Function Frame Model to Understand `countDown`

```
def countDown(n):
    '''Prints ints from n down to 1'''
    if n < 1:
        pass # do nothing
    else:
        print(n)
        countDown(n-1)
```

```
def countDown(n):
    '''Prints ints from n down to 1'''
    if n < 1:
        pass # do nothing
    else:
        print(n)
        countDown(n-1)
```

```
In[1] countDown(5)

 5
 4
 3
 2
 1
```

```
In[2] countDown(4)

4
3
2
1
```

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
 else:
    print(n)
    countDown(n-1)
```

```
>>> countDown(3)
```

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
 else:
    print(n)
    countDown(n-1)
```

```
>>> countDown(3)
```

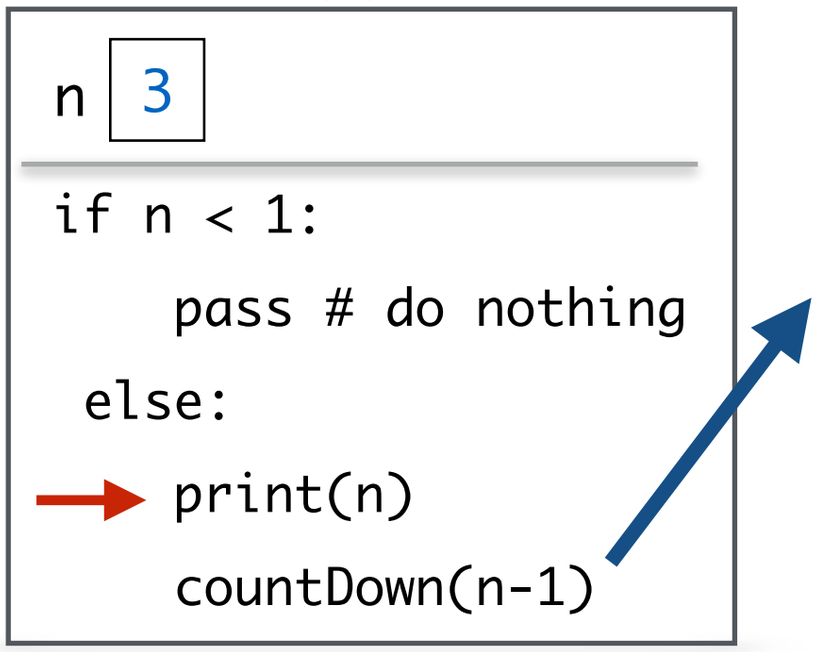
countDown(3)

n 3

```
if n < 1:
    pass # do nothing
 else:
    print(n)
    countDown(n-1)
```

```
>>> countDown(3)
3
```

countDown(3)




```
>>> countDown(3)
3
```

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

```
>>> countDown(3)
3
```

countDown(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

```
>>> countDown(3)
3
```

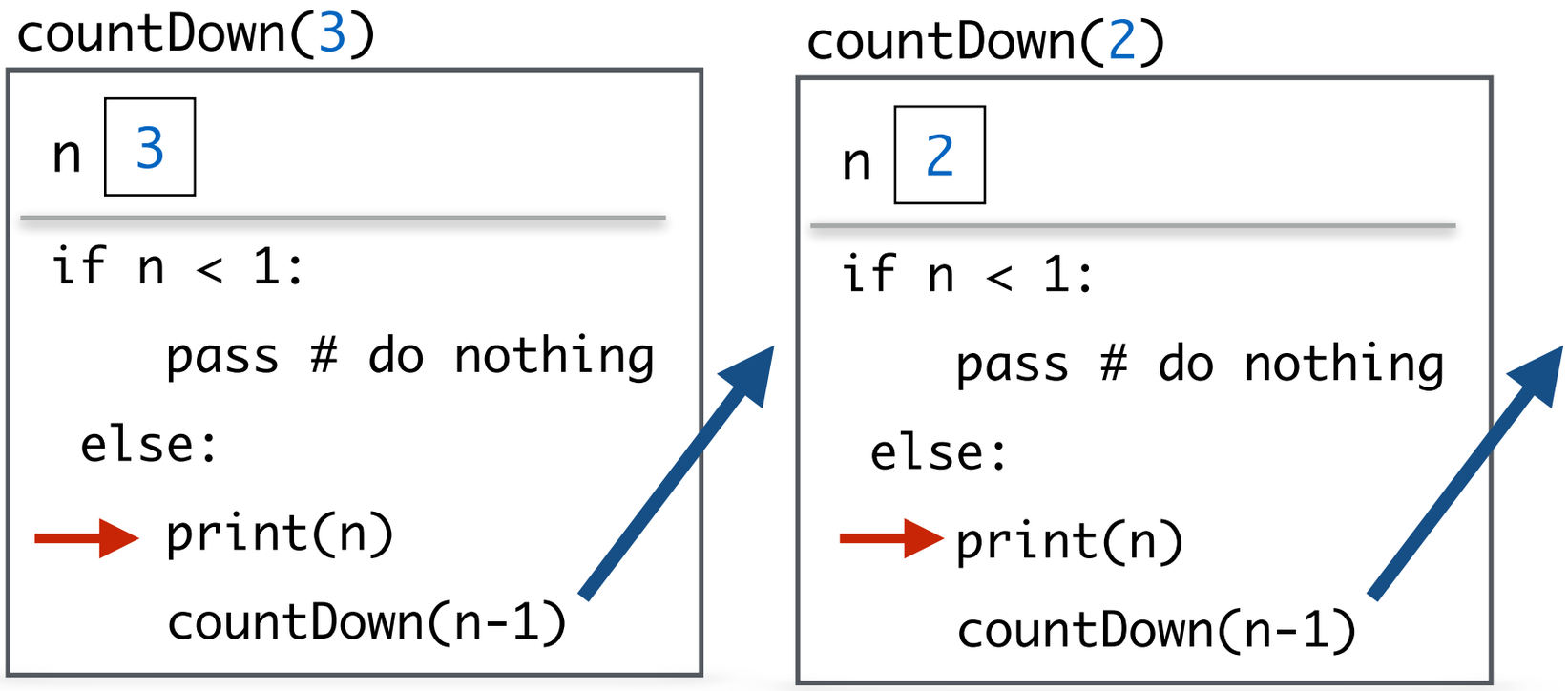
countDown(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

```
>>> countDown(3)
3
2
```




```
>>> countDown(3)
3
2
```

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

```
>>> countDown(3)
3
2
```

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

```
>>> countDown(3)
3
2
```

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

```
>>> countDown(3)
3
2
1
```

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

```
>>> countDown(3)
3
2
1
```

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

```
>>> countDown(3)
3
2
1
```

countDown(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Base case reached!

countDown(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

```
>>> countDown(3)
3
2
1
```

countDown(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Base case reached!

```
>>> countDown(3)
3
2
1
```

countDown(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Base case reached!

countDown(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

```
>>> countDown(3)
3
2
1
```

countDown(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Base case reached!

countDown(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

```
>>> countDown(3)
3
2
1
```

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

Base case reached!

```
>>> countDown(3)
3
2
1
```

countDown(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

Base case reached!

countDown(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

```
>>> countDown(3)
3
2
1
```

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

Base case reached!

countDown(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

```
>>> countDown(3)
3
2
1
```

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

Implicit return

Base case reached!

countDown(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

```
>>> countDown(3)
3
2
1
```

countDown(3)

```
n 3
if n < 1:
    pass # do nothing
else:
→   print(n)
    countDown(n-1)
```

countDown(2)

```
n 2
if n < 1:
    pass # do nothing
else:
→   print(n)
    countDown(n-1)
```

Implicit return

countDown(1)

```
n 1
if n < 1:
    pass # do nothing
else:
→   print(n)
    countDown(n-1)
```

Implicit return

Base case reached!

```
>>> countDown(3)
3
2
1
```

countDown(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

countDown(3)

```
n 3
if n < 1:
    pass # do nothing
else:
→   print(n)
    countDown(n-1)
```

countDown(2)

```
n 2
if n < 1:
    pass # do nothing
else:
→   print(n)
    countDown(n-1)
```

Implicit return

countDown(1)

```
n 1
if n < 1:
    pass # do nothing
else:
→   print(n)
    countDown(n-1)
```

Implicit return

Base case reached!

```
>>> countDown(3)
3
2
1
```

countDown(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

Base case reached!

```
>>> countDown(3)
3
2
1
```

countDown(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

Base case reached!

```
>>> countDown(3)
3
2
1
```

countDown(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

countDown(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

countDown(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

countDown(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

Base case reached!

```
>>> countDown(3)
3
2
1
```

countDown(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    print(n)
    countDown(n-1)
```

Implicit return

More Recursion: countUp

countUp(n)

countUp(n)

- Recursive function that prints integers from 1 up to n (**without using any loops**)

countUp(n)

- Recursive function that prints integers from 1 up to n (**without using any loops**)

```
In[1] countUp(5)

 1
 2
 3
 4
 5
```

```
In[2] countUp(4)

 1
 2
 3
 4
```

Function Frame Model to Understand countUp

countUp(3)

n 3

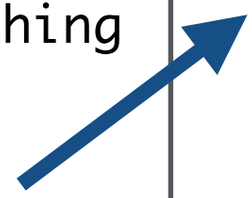
```
if n < 1:
    pass # do nothing
 else:
    countUp(n-1)
    print(n)
```

```
>>> countUp(3)
```

countUp(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```



```
>>> countUp(3)
```

countUp(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

```
>>> countUp(3)
```

countUp(3)

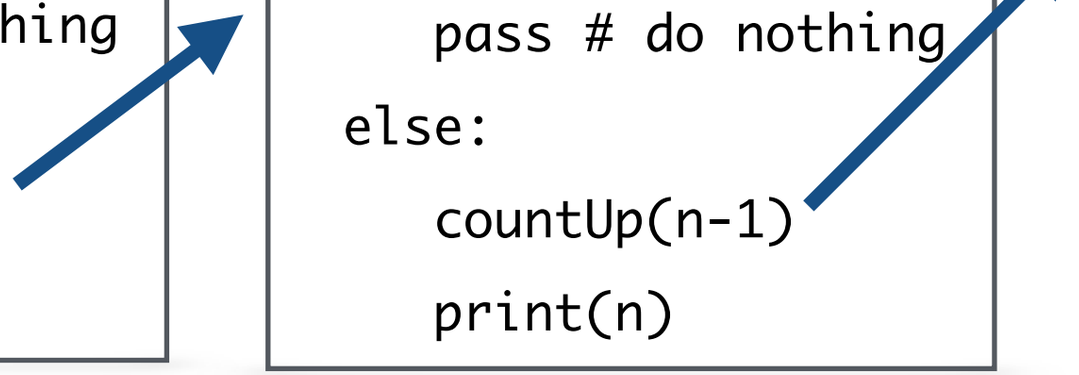
n 3

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```



```
>>> countUp(3)
```

countUp(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

```
>>> countUp(3)
```

countUp(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

```
>>> countUp(3)
```

countUp(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

```
>>> countUp(3)
```

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Base case reached!

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

```
>>> countUp(3)
```

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Base case reached!

```
>>> countUp(3)
```

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Base case reached!

```
>>> countUp(3)
```

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

countUp(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Base case reached!

countUp(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
```

countUp(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Base case reached!

countUp(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
```

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Base case reached!

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
1
```

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Base case reached!

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
1
```

countUp(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Base case reached!

countUp(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
1
```

countUp(3)

n 3

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

n 2

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

n 1

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Base case reached!

countUp(0)

n 0

```
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
1
```

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Base case reached!

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
1
2
```

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

Base case reached!

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
1
2
```

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Base case reached!

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
1
2
```

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→   print(n)
```

Implicit return

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→   print(n)
```

Base case reached!

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
1
2
```

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→ print(n)
```

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→ print(n)
```

Implicit return

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→ print(n)
```

Base case reached!

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
1
2
```

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→ print(n)
```

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→ print(n)
```

Implicit return

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→ print(n)
```

Base case reached!

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
1
2
3
```

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→ print(n)
```

Implicit return

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→ print(n)
```

Implicit return

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→ print(n)
```

Base case reached!

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
1
2
3
```

countUp(3)

```
n 3

if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

countUp(2)

```
n 2

if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

countUp(1)

```
n 1

if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Base case reached!

```
>>> countUp(3)
1
2
3
```

countUp(0)

```
n 0

if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

countUp(3)

```
n 3
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→ print(n)
```

Implicit return

countUp(2)

```
n 2
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→ print(n)
```

Implicit return

countUp(1)

```
n 1
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
→ print(n)
```

Base case reached!

countUp(0)

```
n 0
if n < 1:
    pass # do nothing
else:
    countUp(n-1)
    print(n)
```

Implicit return

```
>>> countUp(3)
1
2
3
```

More Recursion:

`countDownUp`

countDownUp(n)

countDownUp(n)

- Recursive function that prints integers from **n** down to 1 and back up from 1 up to **n** (**without using any loops**)

countDownUp(n)

- Recursive function that prints integers from **n** down to 1 and back up from 1 up to **n** (**without using any loops**)

```
In[1] countDownUp(5)
5
4
3
2
1
1
2
3
4
5
```

countDownUp(n)

- Recursive function that prints integers from **n** down to 1 and back up from 1 up to **n** (**without using any loops**)

```
In[1] countDownUp(5)
5
4
3
2
1
1
2
3
4
5
```

```
In[1] countDownUp(4)

4
3
2
1
1
2
3
4
```

Recursive Patterns

triangle(n, c)

```
>>> triangle(10, '*')
```

```
**********
*********
********
*******
******
*****
****
***
**
*
```

triangleAlt(n, c1, c2)

```
>>> triangleAlt(10, '@', '#')
```

```
@@@@@@@@@@
##########
@@@@@@@@
#######
@@@@@@
#####
@@@@
###
@@
#
```

Function Frame Model:

`triangleAlt(3, '*', '^')`

```
triangleAlt(3, '*', '^')
```

n 3 c1 '*' c2 '^'

```
if n < 1:
    pass # do nothing
 else:
    print(n * c1)
    triangleAlt(n-1, c2, c1)
```

```
>>> triangleAlt(3, '*', '^')
```

```
triangleAlt(3, '*', '^')
```

n 3 c1 '*' c2 '^'

```
if n < 1:
    pass # do nothing
 else:
    print(n * c1)
    triangleAlt(n-1, c2, c1)
```

```
>>> triangleAlt(3, '*', '^')
```

```
triangleAlt(3, '*', '^')
```

n 3 c1 '*' c2 '^'

```
if n < 1:
    pass # do nothing
 else:
    print(n * c1)
    triangleAlt(n-1, c2, c1)
```

```
>>> triangleAlt(3, '*', '^')
***
```

```
triangleAlt(3, '*', '^')
```

n 3 c1 '*' c2 '^'

```
if n < 1:
    pass # do nothing
 else:
    print(n * c1)
    triangleAlt(n-1, c2, c1)
```

```
triangleAlt(2, '*', '^')
```

n 2 c1 '^' c2 '*'

```
if n < 1:
    pass # do nothing
 else:
    print(n * c1)
    triangleAlt(n-1, c2, c1)
```

```
>>> triangleAlt(3, '*', '^')
***
```

```
triangleAlt(3, '*', '^')
```

```
n 3   c1 '*'   c2 '^'
if n < 1:
    pass # do nothing
 else:
    print(n * c1)
    triangleAlt(n-1, c2, c1)
```

```
triangleAlt(2, '*', '^')
```

```
n 2   c1 '^'   c2 '*'
if n < 1:
    pass # do nothing
 else:
    print(n * c1)
    triangleAlt(n-1, c2, c1)
```

```
>>> triangleAlt(3, '*', '^')
***
 ^^
```

```
triangleAlt(3, '*', '^')
```

```
n 3   c1 '*'   c2 '^'
if n < 1:
    pass # do nothing
 else:
    print(n * c1)
    triangleAlt(n-1, c2, c1)
```

```
triangleAlt(2, '*', '
```

```
n 2   c1 '^'   c2 '
if n < 1:
    pass # do nothin
 else:
    print(n * c1)
    triangleAlt(n-1,
```

```
triangleAlt(1, '*', '^')
```

```
n 1   c1 '*'   c2 '^'
if n < 1:
    pass # do nothing
 else:
    print(n * c1)
    triangleAlt(n-1, c2, c1)
```

```
>>> triangleAlt(3, '*', '^')
***
 ^^
```

```
triangleAlt(3, '*', '^')
```

```
n 3   c1 '*'   c2 '^'
if n < 1:
    pass # do nothing
 else:
    print(n * c1)
    triangleAlt(n-1, c2, c1)
```

```
triangleAlt(2, '*', '
```

```
n 2   c1 '^'   c2 '
if n < 1:
    pass # do nothin
 else:
    print(n * c1)
    triangleAlt(n-1,
```

```
triangleAlt(1, '*', '^')
```

```
n 1   c1 '*'   c2 '^'
if n < 1:
    pass # do nothing
 else:
    print(n * c1)
    triangleAlt(n-1, c2, c1)
```

```
>>> triangleAlt(3, '*', '^')
***
 ^^
  *
```

Class Exercise

It's your Turn!

triangleDownUp(n, c1, c2)

```
>>> triangleDownUp(8, '@', '#')
@@@@@@@@
########
@@@@@@
######
@@@@
####
@@
#
#
@@
###
@@@@
#####
@@@@@@
#######
@@@@@@@@
```

Acknowledgments

These slides have been adapted from:

- http://cs111.wellesley.edu/spring19 and
- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/