

CMSC201

Computer Science I for Majors

Lecture 17 – Recursion (cont)

Last Class We Covered

- Recursion
 - Recursion
 - Recursion
- Stacks
- Parts of a recursive function:
 - Base case: when to stop
 - Recursive case: when to go (again)

Any Questions from Last Time?

Today's Objectives

- To gain a more solid understanding of recursion
- To explore what goes on “behind the scenes”
- To examine individual examples of recursion
 - Fibonacci Sequence
- To better understand when it is best to use recursion, and when it is best to use iteration

Review of Recursion

What is Recursion?

- Solving a problem using recursion means the solution depends on solutions to smaller instances of the same problem
- In other words, to define a function or calculate a number by the repeated application of an algorithm

Recursive Procedures

- When creating a recursive procedure, there are a few things we want to keep in mind:
 - We need to break the problem into smaller pieces of itself
 - We need to define a “base case” to stop at
 - The smaller problems we break down into need to eventually reach the base case

“Cases” in Recursion

- A recursive function must have two things:
- At least one base case
 - When a result is returned (or the function ends)
 - “When to stop”
- At least one recursive case
 - When the function is called again with new inputs
 - “When to go (again)”

Code Tracing: Recursion

Stacks and Tracing

- Stacks will help us track what we are doing when tracing through recursive code
- Remember, stacks are **LIFO** data structures
 - Last In, First Out
- We'll be doing a recursive trace of the summation function

Summation Funcion

- The addition of a sequence of numbers
- The summation of a number is that number plus all of the numbers less than it (down to 0)
 - Summation of 5: $5 + 4 + 3 + 2 + 1$
 - Summation of 6: $6 + 5 + 4 + 3 + 2 + 1$
- What does a recursive implementation look like? What's the base case? Recursive case?

Summation Function

```
def summ(num) :  
    if num == 0 :  
        return 0  
    else :  
        return num + summ(num-1)
```

Base case:

Don't want to go below 0
Summation of 0 is 0

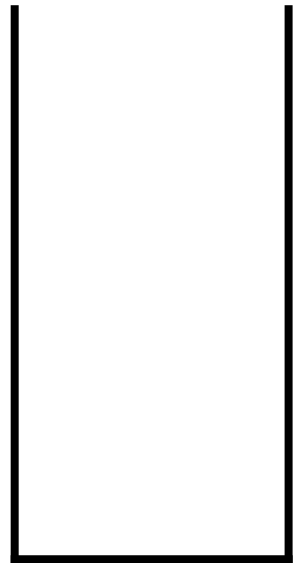
Recursive case:

Otherwise, summation is
num + summation(num-1)

```
main()
```

```
def main():  
    summ(4)
```

```
def summ(num):  
    if num == 0:  
        return 0  
    else:  
        return num + summ(num-1)
```



STACK

```
main()
```



```
def main():
```

```
    ↓ summ(4)
```

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

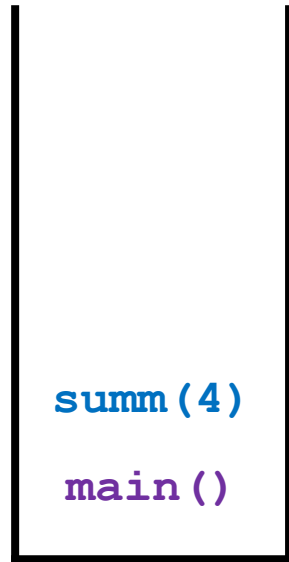
```
    else:
```

```
        return num + summ(num-1)
```

main()

STACK

```
main()
  ↓
def main():
  ↓ summ(4)
    ↓ num = 4
      ↓
def summ(num):
  ↓ if num == 0: num: 4
    return 0
  else:
    return num + summ(num-1)
```



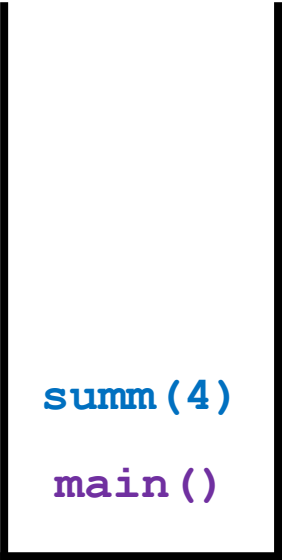
STACK

```

main()
↓
def main():
    ↓ summ(4)
    num = 4
    ↓
def summ(num):
    ↓ if num == 0:
        return 0
    else:
        return num + summ(num-1)
    
```



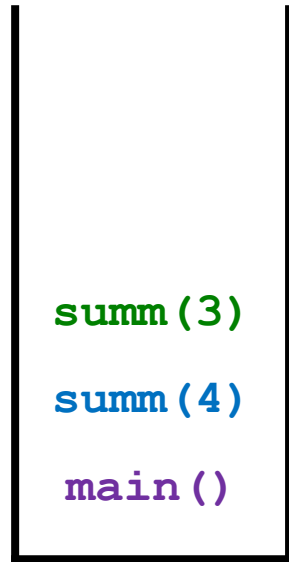
This is a local variable. Each time the `summ()` function is called, the new instance gets its own unique local variables.



STACK


```

main()
  ↓
def main():
  ↓ summ(4)
    num = 4
    ↓
def summ(num):
  ↓ if num == 0: num: 4
    return 0
  else:
    return num + summ(num-1)
    ↓
def summ(num): num = 3
  ↓ if num == 0: num: 3
    return 0
  else:
    return num + summ(num-1)
  
```



STACK

main()

```
def main():
```

```
    summ(4)
```

num = 4

```
def summ(num):
```

```
    if num == 0:
```

num: 4

```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

```
def summ(num):
```

num = 3

```
    if num == 0:
```

num: 3

```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

```
def summ(num):
```

```
    if num == 0:
```

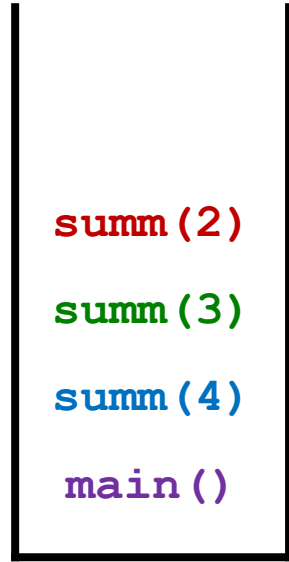
```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

num: 2

2



STACK

main()

```
def main():
```

```
    summ(4)
```

num = 4

```
def summ(num):
```

```
    if num == 0:
```

num: 4

```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

```
def summ(num):
```

num = 3

```
    if num == 0:
```

num: 3

```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

num: 2

```
    else:
```

```
        return num + summ(num-1)
```

num = 1

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

num: 1

```
    else:
```

```
        return num +
            summ(num-1)
```

summ(1)

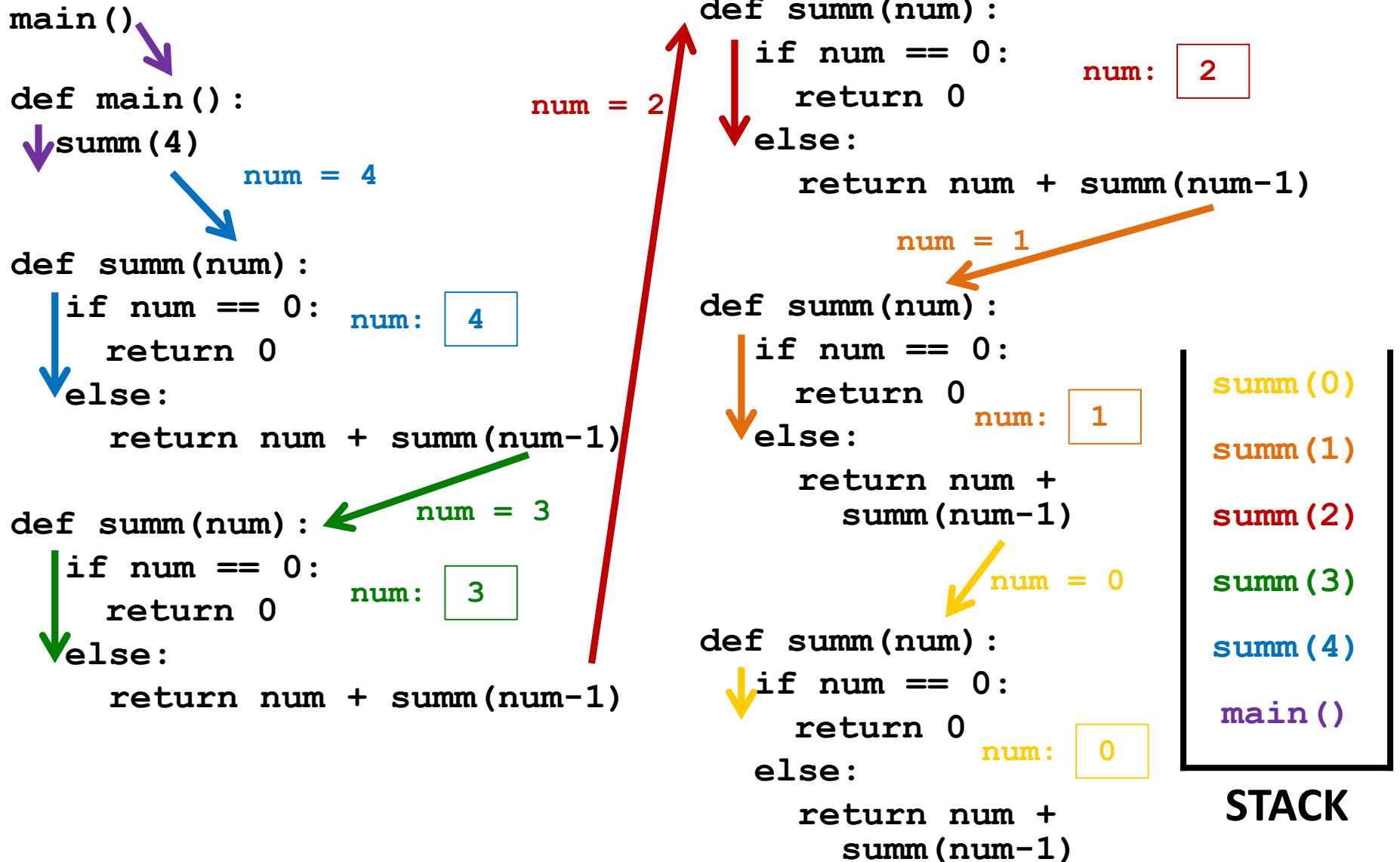
summ(2)

summ(3)

summ(4)

main()

STACK



main()

```
def main():
```

```
    summ(4)
```

num = 4

```
def summ(num):
```

```
    if num == 0:
```

num: 4

```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

```
def summ(num):
```

num = 3

```
    if num == 0:
```

num: 3

```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

num = 2

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

num: 2

```
    else:
```

```
        return num + summ(num-1)
```

num = 1

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

num: 1

```
    else:
```

```
        return num +
            summ(num-1)
```

num = 0

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

num: 0

```
    else:
```

```
        return num +
            summ(num-1)
```

summ(0)

summ(1)

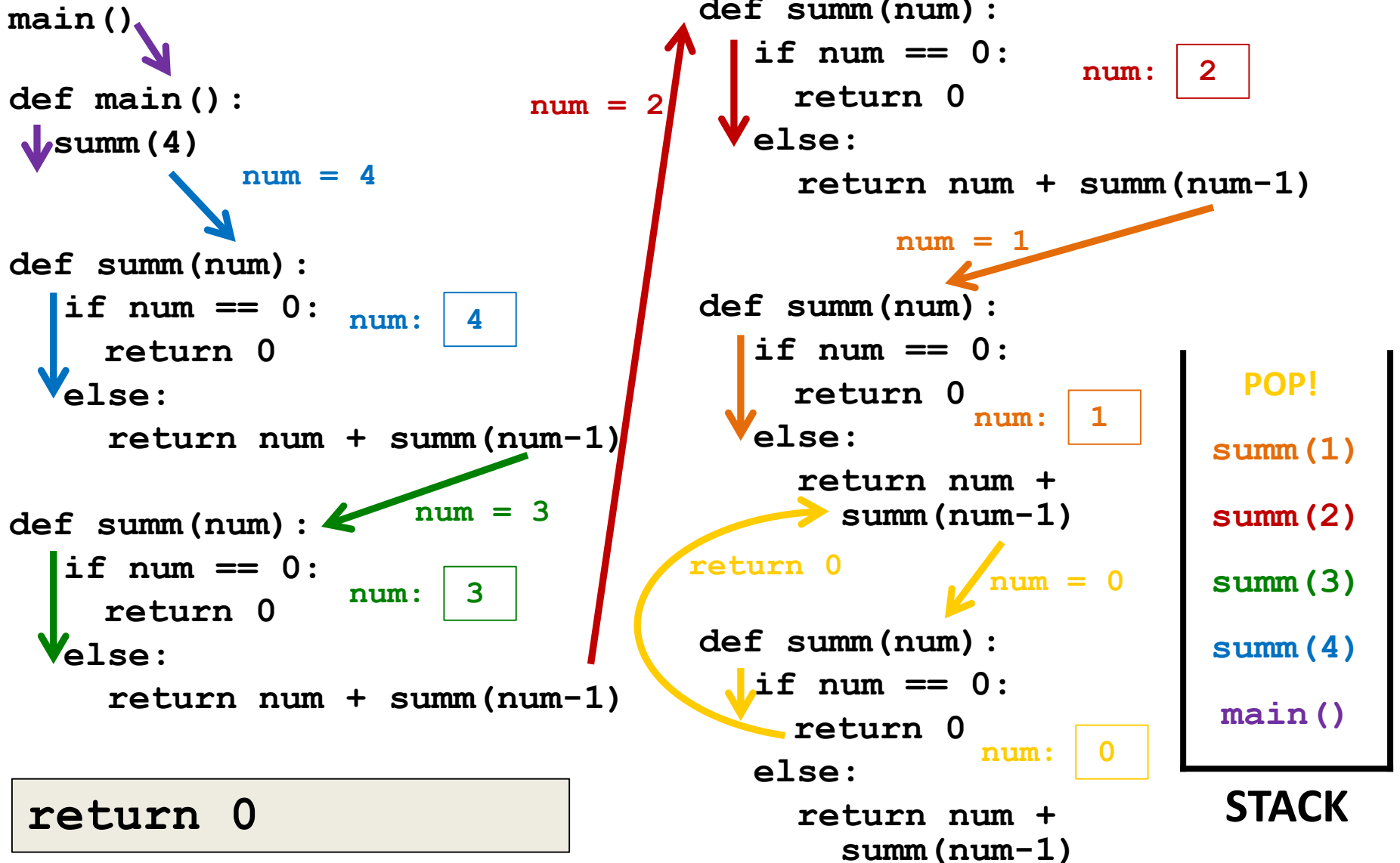
summ(2)

summ(3)

summ(4)

main()

STACK



main()

```
def main():
```

```
    summ(4)
```

num = 4

```
def summ(num):
```

```
    if num == 0:
```

num: 4

```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

```
def summ(num):
```

num = 3

```
    if num == 0:
```

num: 3

```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

num = 2

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

num: 2

num = 1

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

```
    else:
```

```
        return num +
            summ(num-1)
```

num: 1

return 1

POP!

POP!

summ(2)

summ(3)

summ(4)

main()

STACK

return 1 + 0 (= 1)

main()

```
def main():
```

```
    summ(4)
```

num = 4

```
def summ(num):
```

```
    if num == 0:
```

num: 4

```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

```
def summ(num):
```

num = 3

```
    if num == 0:
```

num: 3

```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

num = 2

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

```
    else:
```

```
        return num + summ(num-1)
```

num: 2

return 3

return 2 + 1 (= 3)

POP!

POP!

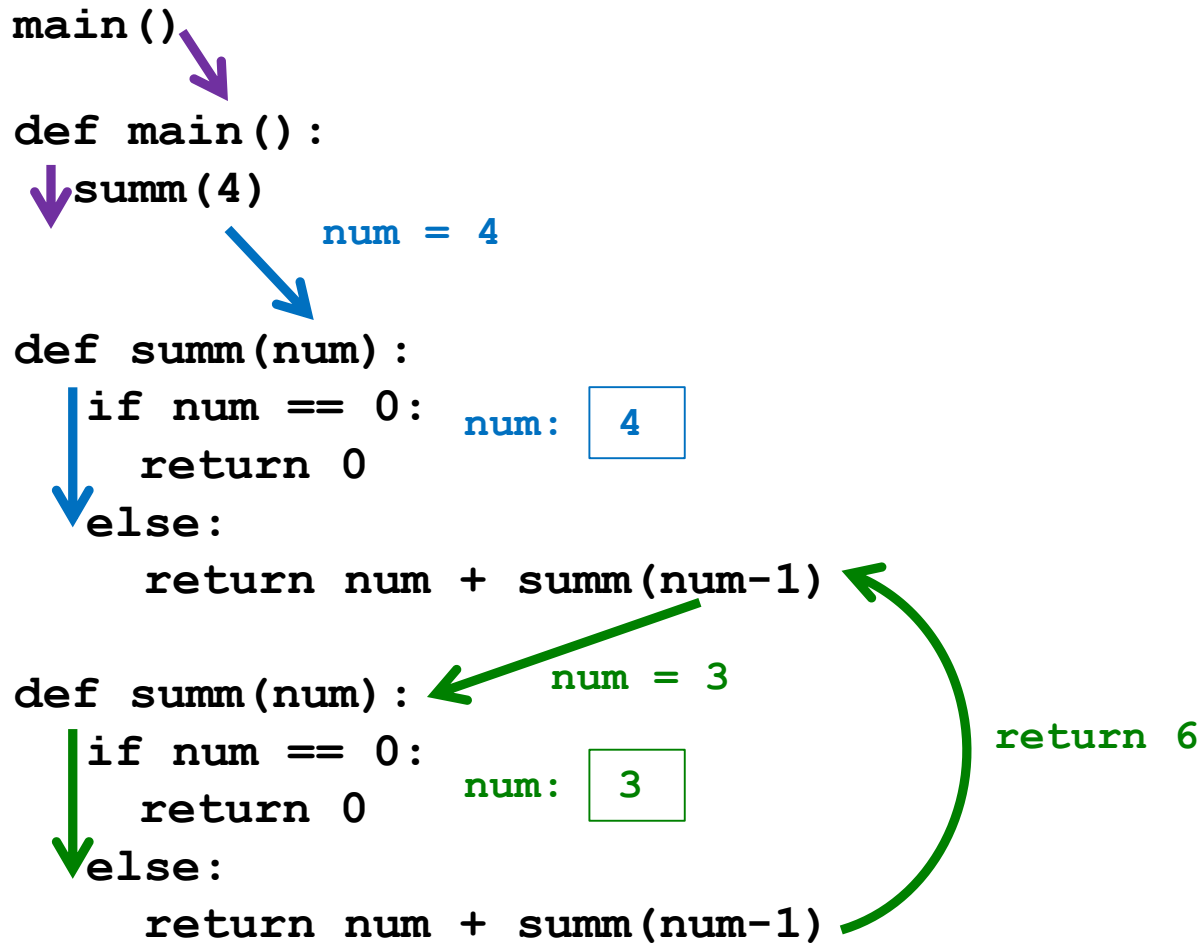
POP!

summ(3)

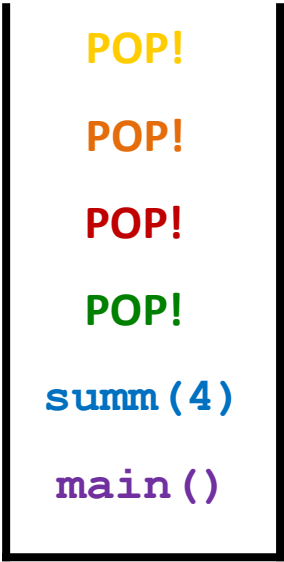
summ(4)

main()

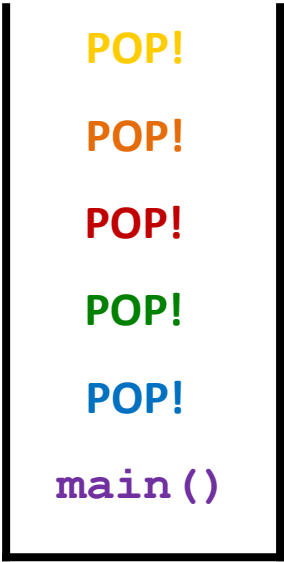
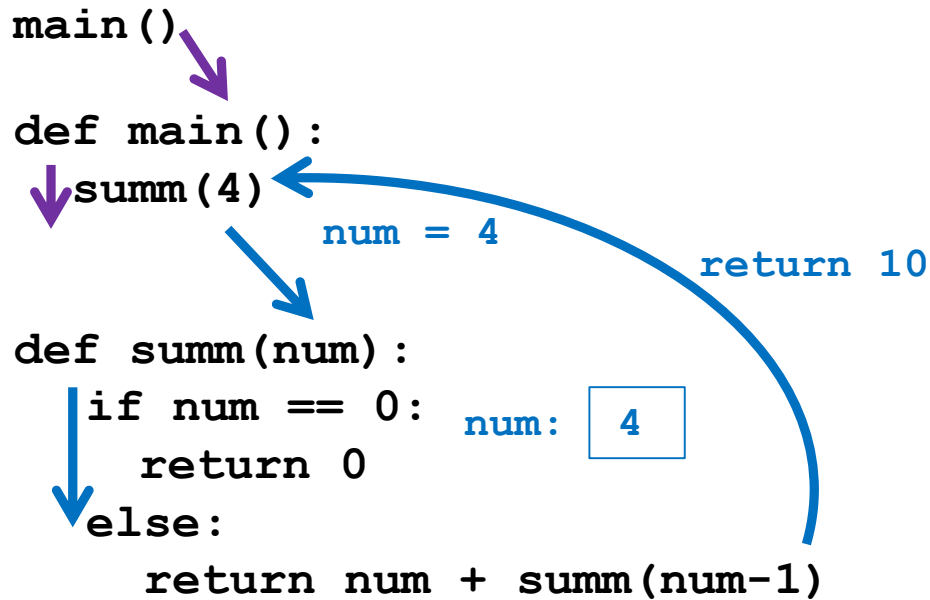
STACK



return 3 + 3 (= 6)



STACK



STACK

return 4 + 6 (=10)

`main()`

`def main():`

`↓ summ(4)`

`return None`

`return None`

POP!

POP!

POP!

POP!

POP!

POP!

STACK

`return control`

The stack is empty!

POP!

POP!

POP!

POP!

POP!

POP!

STACK

Returning and Recursion

Returning Values

- If your goal is to return a final value
 - Every recursive call must return a value
 - You must be able to pass it “back up” to **main()**
 - In most cases, the base case should return as well
- Remember to pay attention to what happens at the “end” of a function

main()

```
def main():
```

```
    summ(4)
```

num = 4

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

num: 4

4

```
    else:
```

```
        num + summ(num-1)
```

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

num: 3

3

```
    else:
```

```
        num + summ(num-1)
```

num = 2

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

num: 2

2

```
    else:
```

```
        num + summ(num-1)
```

num = 1

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

num: 1

1

```
    else:
```

```
        num + summ(num-1)
```

num = 0

```
def summ(num):
```

```
    if num == 0:
```

```
        return 0
```

num: 0

0

```
    else:
```

```
        num + summ(num-1)
```

summ(0)

summ(1)

summ(2)

summ(3)

summ(4)

main()

STACK

Does this work? What's wrong?

Recursion vs Iteration

Recursion and Iteration

- Both are important
 - All modern programming languages support them
 - Some problems are easy to solve when using one and difficult to solve if using the other
- How do you decide which to use?

Use Iteration When...

- Speed and efficiency is an issue
 - Iteration doesn't push things onto the stack
 - Can't "run out" of room like recursion does
- The problem is an obvious fit for iteration
 - Processing every element of a list (or a 2D list)

Use Recursion When...

- Speed is not an issue
- The data being processed is recursive
 - A hierarchical data structure
- A recursive algorithm is obvious
 - (Will happen with time, as you gain experience)
- Clarity and simplicity of code is important

Recursion Practice

Fibonacci Sequences

Fibonacci Sequence

- Number series
- Starts with 0 and 1
- Next number is found by adding the previous two numbers together
- Pattern is repeated over and over (and over...)

Fibonacci Sequence

- Starts with 0, 1, 1
- Next number is ...?

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

Time for...

LIVECODING!!!

Recursively Implement Fibonacci

- The formula for a position in the sequence:

$$\mathbf{fib}(p) = \mathbf{fib}(p-1) + \mathbf{fib}(p-2)$$

- What is our base case?
- What is our recursive case?

Non-Recursive `exp()`

- How do we create an exponentiation without using the exponentiation operator?
 - Given **number** and **power**, we can use code like this:

```
ans    = 1
count  = 1
while count <= power:
    ans *= number
    count += 1
```

Transform
this into a
recursive
function

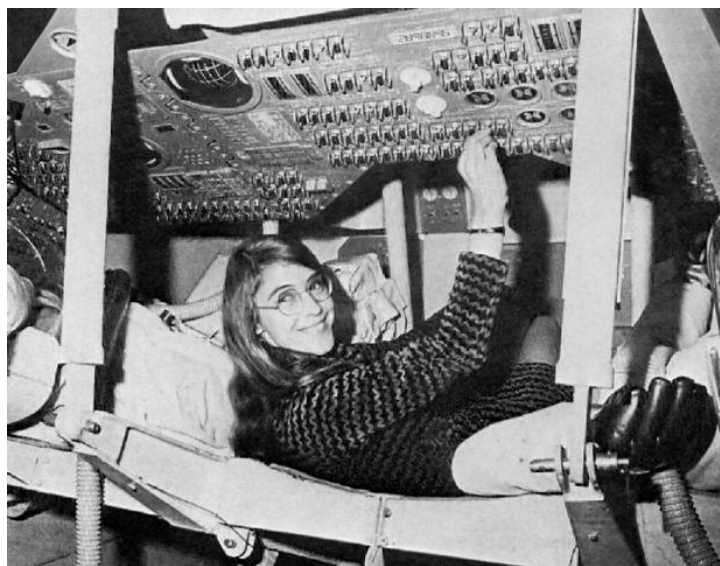
Recursive Answer

```
def recExp (number, power) :  
    # BASE CASE  
    if power == 0:  
        return 1  
    # RECURSIVE CASE  
    else:  
        return number * \  
            recExp (number, power - 1)
```

– There are other correct answers; this is just one

Daily CS History

- Margaret Hamilton
 - Created software for space flight!
 - Apollo 8
 - First to leave orbit
 - Apollo 11
 - Moon landing
 - Invented the term “software engineering”
 - Her daughter got to play with space flight simulators!



Announcements

- Project 2 is out on Blackboard now
 - *Project* is due by Friday (Nov 10th) at 8:59:59 PM
- Homework 6 will come out Saturday
 - The topic is recursion – recommended that you do the assignment parts in order
- Final exam is when?
- Friday, December 15th from 6 to 8 PM

Image Sources

- Margaret Hamilton
 - https://en.wikipedia.org/wiki/File:Margaret_Hamilton_in_action.jpg