# Lect 8 – Recursion

Rob Capra
INLS 490-172

# Recursion

- A method of solving problems by breaking them down into smaller and smaller subproblems until you get a small enough problem that it can be solved trivially.
- Usually involves a function calling itself

# Example #1

- Print a countdown from 10 to 0

```
def countdown(n):
    for i in range(n,0,-1):
        print i
    print "Blastoff!"



countdown(10)
```

# Example #1

- Print a countdown from 10 to 0

```
def countdown(n):
    for i in range(n,0,-1):
        print i
    print "Blastoff!"


countdown(10)
```

```
def countdown2(n):
    if n <= 0:
        print "Blastoff!"
    else:
        print n
        countdown(n-1)

countdown2(10)
```

# Essential Elements of Recursion

- A recursive algorithm must
  - have a base case
  - change its state and move toward the base case
  - call itself, recursively

# Base Case

- The base case is what allows the algorithm to stop recursing
- It should represent a case that is *trivial*
  - Meaning that it cannot be decomposed further and that the solution (to the base case) is simple
- Examples:
  - A list of length 1 is always is sorted order
  - The sum of elements in a list of length 1 is just the value of the one element.

# Change state to move toward base case

- The algorithm should, for each recursive call, move closer to the base case
- Examples
  - A list shrinks by one
  - A number is divided by some factor

# Function should call itself, recursively

- If the function is not in the base case
  - It will typically perform some operation
  - And call itself with an argument that moves things closer to the base case.

# Recursion is a form of iteration

# Be careful of infinite recursion!

```
def neverending(t):
    return neverending(t)
```

# Example #2

- Definition of factorial

  0! = 1

  n! = n*(n-1)!

```
print fac(4)      # ans = 4 * 3 * 2 * 1
24
```

- Think:
  - What should be the base case?
  - How do we progress toward the base case?
  - How will we call the function recursively?

# Example #3

- Fibonacci

  fib(0) = 0

  Fib(1) = 1

  fib(n) = fib(n-1) + fib(n-2)

```
print fib(4)
3
```

fib(4) = fib(3) + fib(2)
= (fib(2) + fib(1)) + (fib(1) + fib(0))
= ((fib(1) + fib(0)) + fib(1)) + (fib(1) + fib(0))
= 1 + 0 + 1 + 1 + 0
= 3

# Example #4

- Sum a list of numbers

  ```
  t = [1, 3, 5, 7, 9]
  ```

- First, let’s do this with a loop
  - (collaboratively write code to do this)

# Example #4

- Sum a list of numbers

  ```
  t = [1, 3, 5, 7, 9]
  ```

- Now let’s try to write a recursive solution
- Think:
  - What should be the base case?
  - How do we progress toward the base case?
  - How will we call the function recursively?

# Example #4

- Sum a list of numbers

```
t = [1, 3, 5, 7, 9]
```

- Idea:

listsum(t) = first(t) + listsum(all_except_first(t))

total = (1 + (3 + (5 + (7 + 9))))

= (1 + (3 + (5 + 16)))

= (1 + (3 + 21))

= (1 + 24)

= 25

# Stack Frames

```
def listsum(t):
    if len(t) == 1:
        return t[0]
    else:
        return t[0] + listsum(t[1:])
```

```
listsum([1, 3, 5, 7])
      1 + listsum([3, 5, 7])
            1 + 3 + listsum([5, 7])
                  1 + 3 + 5 + listsum([7])
                  1 + 3 + 5 + 7
            1 + 3 + 12
      1 + 15
16
```

# Illustration of Recursive Calls

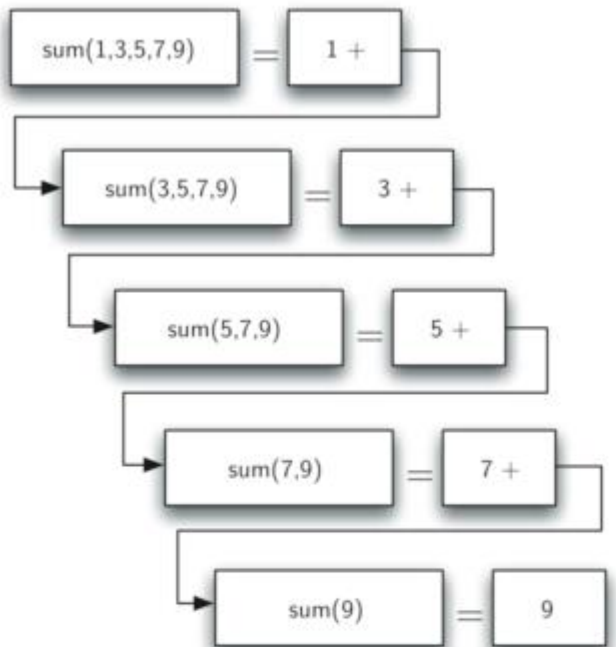


Figure 1: Series of Recursive Calls Adding a List of Numbers

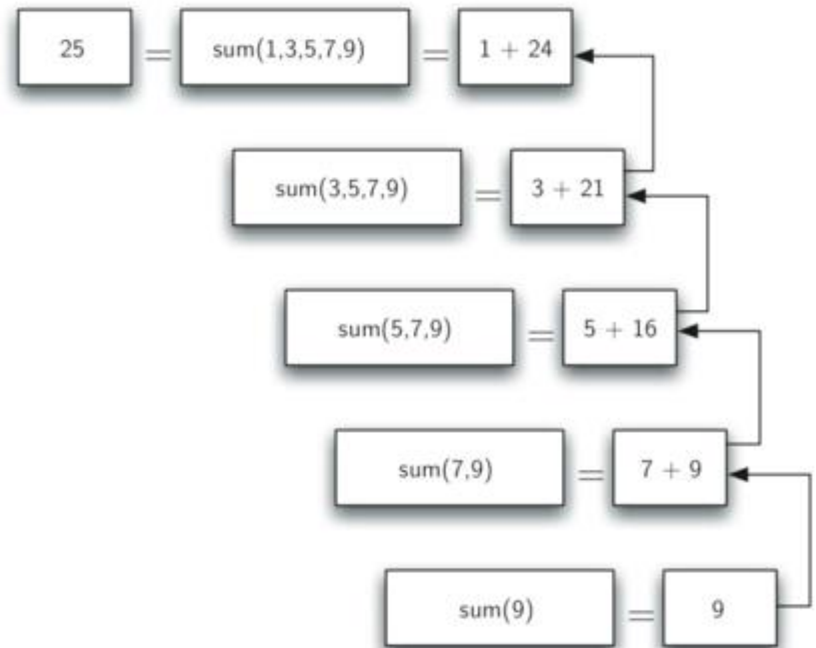


Figure2: Series of Recursive Returns from Adding a List of Numbers

# Example #5

- Reverse a string

```
print str_reverse('python')
nohtyp
```

- First, let's do this with a loop
  - (collaboratively write code to do this)