

CS 5 ... Today!

MAGIC!?

Recursion

As close as CS gets to magic

a.k.a., CS's version of mathematical induction

Tutoring hours: LOTS!

Read Sections 2.7-2.12

This is the *last* CS 5 lecture you'll ever "need"!*

HMC's legal counsel requires us to include these footnotes...

* At Warner Brothers' insistence, we affirm that this 'C' does not stand for 'Chamber' and 'S' does not stand for 'Secrets.'

* **Caution:** do not take this statement too literally or you might find yourself in *twice* as many CS 5 lectures as you need!

Use variables!



I'm happy about this, too!

```
def flipside(s):  
    mid = len(s) // 2  
    return s[mid:] + s[:mid]
```

```
def flipside(s):  
    return s[len(s) // 2:] + s[:len(s) // 2]
```

Aargh!

Why would computers "prefer" the top version, too?

Python is... **in**

I guess Python's the in thing



```
>>> 'i' in 'team'  
False
```

```
>>> 'cs' in 'physics'  
True
```

```
>>> 'i' in 'alien'  
True
```

```
>>> 42 in [41, 42, 43]  
True
```

```
>>> 3*'i' in 'alien'  
False
```

```
>>> 42 in [[42], '42']  
False
```

return vs. **print**

```
def dbl(x):  
    """dble x?"""  
    return 2*x
```

```
In[1]: ans = dbl(20)
```

```
def dblPR(x):  
    """dble x?"""  
    print(2*x)
```

```
In[1]: ans = dblPR(20)
```

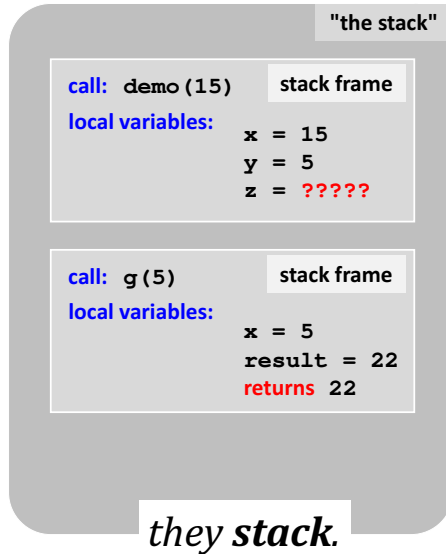
What's the difference ?!

How functions work...

```

15
↓
def demo(x):
    y = x // 3
    z = g(y)
    return z + y + x

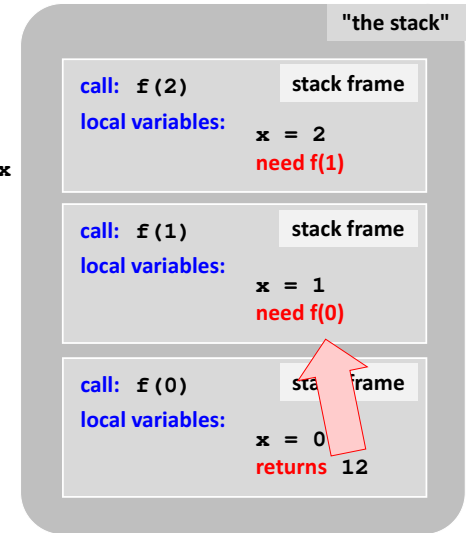
def g(x):
    result = 4*x + 2
    return result
    
```



How functions work...

```

0
↓
def f(x):
    if x == 0:
        return 12
    else:
        return f(x - 1) + 10*x
    
```



Thinking *recursively*

factorial

$$5! = 120$$

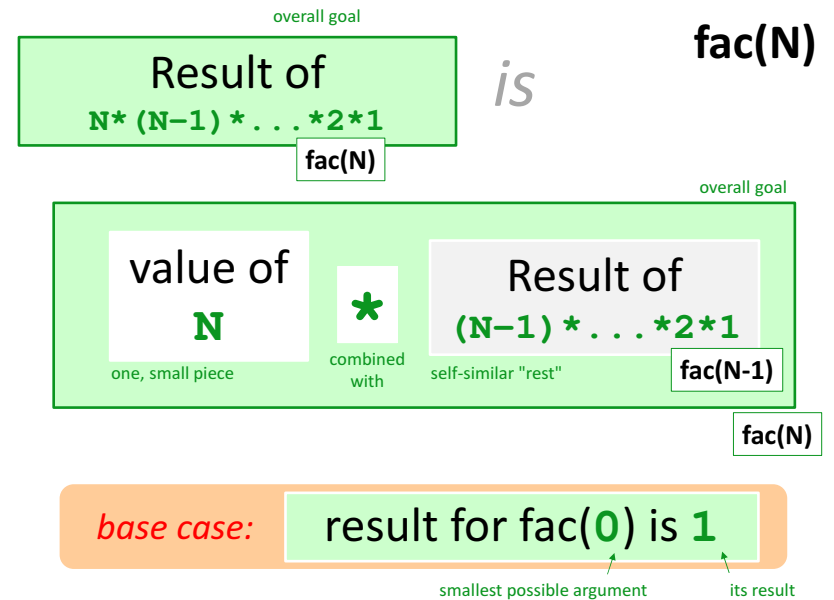
$$\text{fac}(5) = 5 * 4 * 3 * 2 * 1$$

$$\text{fac}(5) =$$

can we express **fac** w/ a smaller version of itself?

$$\text{fac}(N) = N * (N-1) * \dots * 3 * 2 * 1$$

$$\text{fac}(N) =$$



Thinking recursively...

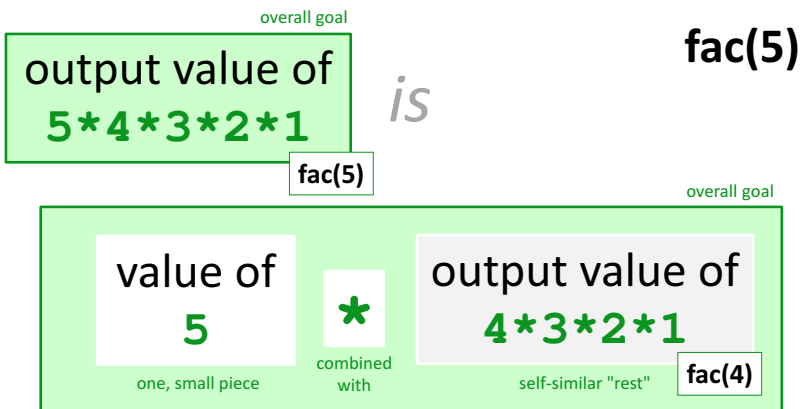
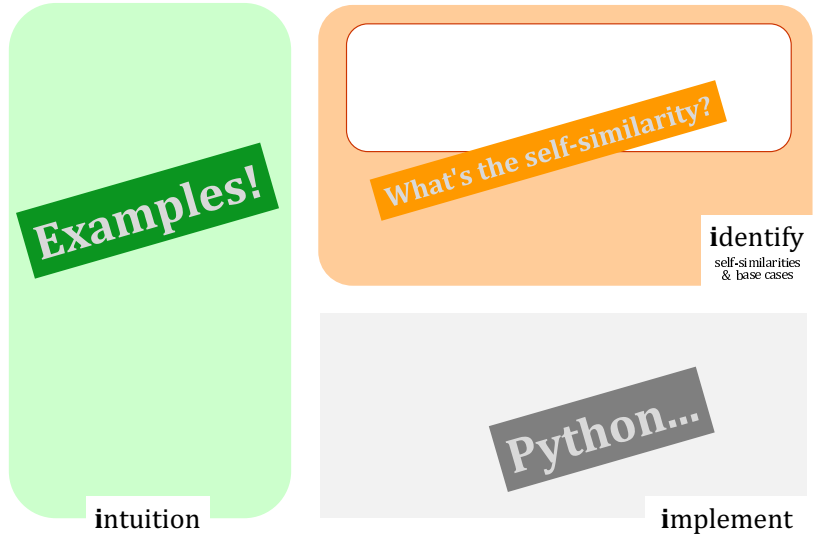
```
def fac(N):
    if N == 0:
        return 1
    else:
        return N * fac(N-1)
```

Base case

Recursive case
(too short?)

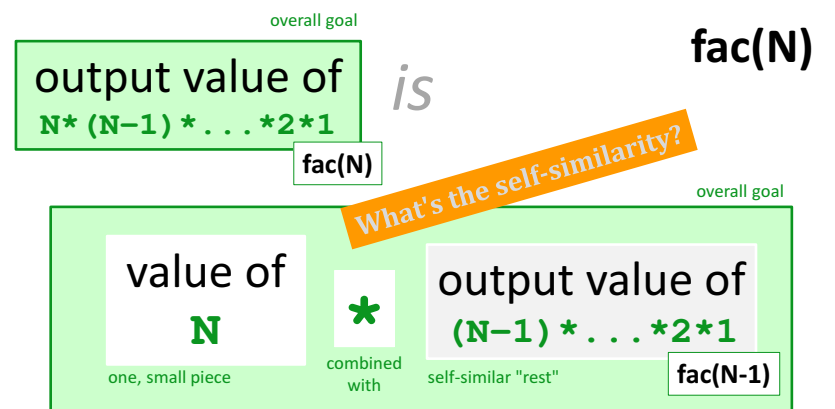


Recursive Design: the 3 i's

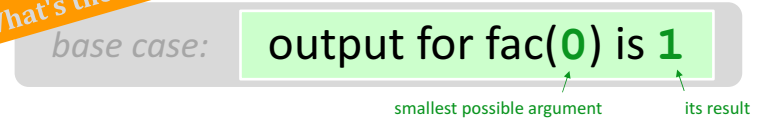


Examples!

What's the self-similarity?



What's the base case?



Recursive Design: the 3 i's



fac(5) is 5*4*3*2*1
 rec.
 fac(4) is 4*3*2*1
 fac(3) is 3*2*1
 fac(2) is 2*1
 fac(1) is 1
 fac(0) is 1
 base case(s)
 intuit

rec.
 fac(N) is $N * \text{fac}(N-1)$
 self-similarities
 base cases
 identify self-similarities & base cases

```

def fac(N):
    if N <= 1:
        return 1.0
    else:
        return N * fac(N-1)
    
```

 base case
 rec.
 implement

Recursive Design: the 3 i's

plusone(4) is 1+1+1+1
 recursion?
 plusone(3) is 1+1+1
 plusone(2) is 1+1
 plusone(1) is 1
 plusone(0) is 0
 base case(s)
 intuit

rec.
 plusone(N)
 self-similarities
 base cases
 identify self-similarities & base cases

```

def plusone(N):
    if ____:
        return ____
    else:
        return ____
    
```

 base case
 rec.
 implement

Recursive Design: the 3 i's

plusone(4) is 1+1+1+1
 recursion?
 plusone(3) is 1+1+1
 plusone(2) is 1+1
 plusone(1) is 1
 plusone(0) is 0
 base case(s)
 intuit

rec.
 plusone(N)
 self-similarities
 base cases
 identify self-similarities & base cases

```

def plusone(N):
    if ____:
        return ____
    else:
        return ____
    
```

 base case
 rec.
 implement

Recursive Design: leng(L)

leng([5, 42, 7]) is 3
 recursion?
 leng([42, 7]) is 2
 leng([7]) is 1
 leng([]) is 0
 base case(s)
 intuit

rec.
 leng(L)
 self-similarities
 base cases
 identify self-similarities & base cases

```

def leng(L):
    if ____:
        return ____
    else:
        return ____
    
```

 base case
 rec.
 implement

Design patterns...

Recursion's a design—not a formula, BUT, these pieces are common:

- Handle base cases, with **if** ...
- Do one piece of work: **L[0]** or **s[0]**
- Recur with the rest: **L[1:]** or **s[1:]**
- Combine & make sure the types match!

pow(b, p)

pow(b, p) returns b**p, but using only multiplication times b



$$\text{pow}(2, 4) \sim \frac{2*2*2*2}{=16}$$

$$\text{pow}(2, 3) \sim \frac{2*2*2}{=8}$$

$$\text{pow}(2, 2) \sim 2*2$$

$$\text{pow}(2, 1) \sim 2$$

$$\text{pow}(2, 0) \sim 1$$

anything to the zero power is 1.0

intuit

pow(b, p)

self-similarities

base cases

identify self-similarities & base cases

```
def pow(b, p):
    if ____:
        return ____
    else:
        return _____
```

Extra! See if you can also handle *negative* powers...

implement

vwl(s)

vwl(s) returns the number of vowels in the string s, s may or may not be empty.



let's count!

vwl('crazy') ~ 2

vwl('razy') ~ 2

vwl('azy') ~ 2

vwl('zy') ~ 1

vwl('y') ~ 1

vwl('') ~ 0

What is the really simplest base case?

intuit

vwl(s)

IF

IF

self-similarities

multiple possibilities!

base cases

identify self-similarities & base cases

```
def vwl(s):
    if ____:
        ____
    elif ____:
        ____
    else:
        ____
```

base case

Extra! What 7-letter English word *maximizes* vwl(s)?

implement

mymax(L)

mymax(L) return the biggest element in L, given that L has at least one element!



mymax([9, 8, 5]) ~ 9

mymax([8, 5]) ~ 8

mymax([2, 8, 5]) ~ 8

mymax([8, 5]) ~ 8

mymax([5]) ~ 5

intuit

mymax(L)

self-similarities

base cases

identify self-similarities & base cases

```
def mymax(L):
    ____
```

implement

zeroest(L)

zeroest (L)

return the element of L closest to 0;
L will have *at least one element*

Consider these examples:

zeroest([4, -3, 42]) ~ -3

zeroest([1, -3, 42]) ~ 1

zeroest([-3, 42]) ~ -3

zeroest([42]) ~ 42

don't forget the
base case!

intuit

zeroest(L)

self-similarities

base cases

identify
self-similarities
& base cases

```
def zeroest (L) :
```

Note: The function `abs` is built into Python....

implement

Quiz

How f'uns *work*...

Name(s):

15



What is `demo(15)` here?

```
def demo(x):  
    y = x // 3  
    z = g(y)  
    return z + y + x
```

```
def g(x):  
    result = 4*x + 2  
    return result
```

I might have a
guess at both
of these...



What is `f(2)` here?

```
def f(x):  
    if x == 0:  
        return 12  
    else:  
        return f(x - 1) + 10 * x
```