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

On 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 it is possible find yourself in twice as many CS 5 lectures as you need!

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

* Footnote: do not take this statement too literally or it is possible find yourself in twice as many CS 5 lectures as you need!

Hw #1 due this Monday, 9/17, at 11:59 pm

This is the last CS 5 lecture you’ll ever “need”!*

Computation's Dual Identity

accessed through functions…

Computation

Data Storage

name: x
type: int
LOC: 300
memory location 300

name: y
type: int
LOC: 304
memory location 304

variables ~ boxes

functions!

It’s no coincidence this starts with fun!

Functioning across disciplines

procedure

structure

CS’s googolizer

Math’s googolizer

defined by what it does

+ what follows behaviorally

defined by what it is

+ what follows logically

def g(x):
    return x**100

$g(x) = x^{100}$
Giving names to data *helps f'ns*

```python
def flipside(s):
    """
    flipside(s): swaps s's sides!
    input s: a string
    """
    x = len(s) // 2
    return s[x:] + s[:x]
```

This idea is the key to your happiness!

Test!

```python
# Tests!
assert flipside('homework') == 'workhome'
assert flipside('poptart') == 'tartpop'
print(" petscar ~", flipside('carpets'))
print(" cs5! ~", flipside('5!cs'))
```

We provide tests *for now...*
**Redefining variables...**

```python
def convertFromSeconds(s):
    # total seconds
    """
    convertFromSeconds(s): Converts an integer # of seconds into a list of
    [days, hours, minutes, seconds]
    input s: an int
    """
    days = s // (24*60*60)  # total days
    s = s % (24*60*60)      # remainder s
    hours = s // (60*60)    # total hours
    s = s % (60*60)        # remainder s
    minutes = s // 60       # total minutes
    s = s % 60             # remainder s
    return [days, hours, minutes, s]
```

**Naming things!**

```python
def convertFromSeconds(s):
    """
    convertFromSeconds(s): Converts an integer # of seconds into a list of
    [days, hours, minutes, seconds]
    input s: an int
    """
    days = s // (24*60*60)  # total days
    s = s % (24*60*60)      # remainder s
    hours = s // (60*60)    # total hours
    s = s % (60*60)        # remainder s
    minutes = s // 60       # total minutes
    s = s % 60             # remainder s
    return [days, hours, minutes, s]
```

**Return vs. Print**

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

ans = dbl(20)
```

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

ans = dblPR(20)
```

What's the difference ?!
def demo(x):
    y = x/3
    z = g(y)
    return z + y + x

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

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

What is demo(15) here?

What is f(2) here?

Thinking **sequentially**

factorial

\[ 5! = 120 \]

\[ \text{cs} \quad \text{fac}(5) = 5 \times 4 \times 3 \times 2 \times 1 \]

\[ \text{fac}(N) = N \times (N-1) \times \ldots \times 3 \times 2 \times 1 \]

Thinking **recursively**

factorial

\[ 5! = 120 \]

\[ \text{cs} \quad \text{fac}(5) = \]

\[ \text{fac}(N) = N \times (N-1) \times \ldots \times 3 \times 2 \times 1 \]

\[ \text{fac}(N) = \]

can we express \text{fac} w/ a smaller version of itself?

**Quiz**

How fns work...

sequential
text iteration

text self-similar
text recursion

problem-solving **paradigms**
Warning: **this is legal!**

```python
def fac(N):
    return N * fac(N-1)
```

I wonder how this code will STACK up!? 😋

```python
def facBad(N):
    return N * facBad(N-1)
```

**legal != recommended**

```python
def facBad(N):
    return N * facBad(N-1)
```

Calls to `facBad` will "never" stop: there's no **base case**

Make sure you have a **base case**

a.k.a. "escape hatch"

---

**Thinking recursively...**

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

**Acting recursively**

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

**Conceptual**

**Actual**

---

Thinking recursively...

Acting recursively
Behind the curtain: how recursion works...

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

Recursion's conceptual challenge?
You need to see BOTH the self-similar pieces AND the whole thing simultaneously!

Nature loves recursion!
... because it's completely self-sufficient!

Recursion

Base Case

Self-similar design

problem-solving paradigm

Recursion

Base Case

Self-similar design

Base case:
fac(0) should return 1

fac(5)

value of
5*4*3*2*1

is

fac(4)

value of 5 *
4*2*3*1

fac(0)

factorial of x
def fac(x):
    """ factorial! Recursively! ""
    if x == 0:
        return 1
    else:
        return x*fac(x-1)

def pow(b, p):
    """ b**p, defined recursively! ""
    if p == 0:
        return __________
    else:
        return __________

Recursion's advantage: It handles arbitrary structural depth – all at once + on its own!

As a hat, I'm recursive, too!
Recursion’s a design - not a formula, but these pieces are common:

Design patterns...

\[ s = '\text{aliienen}' \]

in terms of \( s \), what are these pieces? (index! slice!)

Design patterns...

\[ \text{\texttt{\# of i's in 'xlii'}} \quad \text{\texttt{\# of i's in 'x'}} + \text{\texttt{\# of i's in 'lii'}} \]

Base case: `numis('')` should return ___?

```
def numis(s):
    
    if s == '':
        return 

    elif s[0] == 'i':
        return 

    else:
        return 
```
**Leap before you look!**

Try these four...

Base case:

len('') should return ___ ?

def len(s):
    
    # returns the length of s
    
    if s == '':
        return _____

    else:
        return __________

Base case:
vwl('') should return ____ ?

vwl('eerie')

# of vowels in 'eerie'

# of vowels in 'e' + # of vowels in 'erie'

Base case:
vwl('') should return ____ ?
```python
def vwl(s):
    
    
    
    
    def keepvwl(s):
        
        
        
        
        if s == '':
            return ________
        elif s[0] in 'aeiou':
            return ______________
        else:
            return ______________
```
**max(L)**

max of

\([7, 5, 9, 2]\]

is

either 7

or the max of

\([5, 9, 2]\]

**zeroest(L)**

zeroest of

\([-7, 5, 9, 2]\]

is

either -7

or the zeroest of

\([5, 9, 2]\]

---

**Base case:**

if len(L) == 1, what should max(L) return?

**def max(L):**

    """ returns the max of L! ""

    if len(L) == 1:
        return 

    M = 

    if : 
        return 
    else:
        return 

**The max of the REST of L**

**Base case:**

if len(L) == 1, what should zeroest(L) return?

**def zeroest(L):**

    """ returns L's element nearest 0 ""

    if len(L) == 1:
        return 

    Z = 

    if : 
        return 
    else:
        return 

**The zeroest of the REST of L**
The key to understanding recursion is, first, to understand recursion.

- former CS 5 student

Good luck with Homework #2

It's the eeriest!

tutors @ LAC  Th/F/Sa/Su/Mon.