We're computationally complete!

putting Python to work! & adding building blocks

---

**Functional Programming**

>>> 'fun' in 'functional'

True

- Representation via list structures (data)
- Leverage self-similarity (recursion)
- Create small building blocks (functions)

**Composed together** — to solve/investigate problems

**Functional programming**

conceptually concise vs. easiest for the computer...

functional procedural or sequential

---

```python
def sum(L):
    """Argument: L, a list of numbers
    Result: L's sum
    """
    if len(L) == 0:
        return 0
    else:
        return L[0] + sum(L[1:])

def range(low, high):
    """Arguments: ints low and high
    Result: int list from low to high (excluding hi)
    """
    if low >= high:
        return []
    else:
        return list(range(low, hi))
```

---

**CS 5: now recurring...**

Or re-cursing, depending on your feelings about recursion!

Hw3—due Monday evening—usual time

Read Sections 3.6-3.9

---

Putting Python to work! & adding building blocks

---

# Functional Programming

```python
def sum(L):
    """Argument: L, a list of numbers
    Result: L's sum
    """
    if len(L) == 0:
        return 0
    else:
        return L[0] + sum(L[1:])

def range(low, high):
    """Arguments: ints low and high
    Result: int list from low to high (excluding hi)
    """
    if low >= high:
        return []
    else:
        return list(range(low, hi))
```
List Comprehensions

Expression to evaluate for each list element
Name for each list element
The list - or string to run

$[2\times x \text{ for } x \text{ in } [0, 1, 2, 3, 4, 5]]$

$[0, 2, 4, 6, 8, 10]$ result

What's the syntax saying here?

List Comprehensions

$[10\times x \text{ for } x \text{ in } [0, 1, 2, 3, 4, 5] \text{ if } x\%2 == 0]$

$[y\times 21 \text{ for } y \text{ in } \text{list}(\text{range}(0, 3))]$

$[s[1] \text{ for } s \text{ in } ["hi", "5Cs!"] ]$

List Comprehensions

What to do
$x$ takes on each value
this "runner" variable can have any name...

$[2\times x \text{ for } x \text{ in } [0, 1, 2, 3, 4, 5]]$

and $2\times x$ is result for each one

$[0, 2, 4, 6, 8, 10]$ output

LCs for Monte Carlo Analysis...

# this line runs guess(42) 1000 times
LC = [countGuess(42) for x in range(1000)]

# Let's look at the first ten of them:
print(LC[0:10])

# Let's find the average:
print("av. #guesses: ", sum(LC) / len(LC))

a.k.a. Run it a "zillion" times!
Zillion-times testing!

# this runs the doubles-counter 600 times...
cd_np(600)    # np: no printing

# Run _that_ 1000 times (600,000 rolls total!)
LC = [cd_np(600) for x in range(1000)]

# Look at the first 10 of these
print(LC[0:10])

# Then, find the average:
print("avg. dbls (/600):", sum(LC) / len(LC))

---

Designing with LCs

write {_______ for x in range(4)}
[0, 1, 2, 3]

result [0, 14, 28, 42]

write {___________ for c in 'igetthis'}
result [True, False, False, False, False, False, True, False]
[ 1,    0,     0,     0,     0,     0,     1,    0  ]

And if we wanted the ints (in red)...?

---

Using LCs

def fun1(L):
    LC = [1 for x in L]
    return sum(LC)

def letScore(c):
    lots of ifs...
    return score

def fun2(S):
    LC = [letScore(c) for c in S]
    return sum(LC)

def vwl(s):
    LC = [for c in s]
    return sum(LC)

def count(e, L):
    LC = [for x in L]
    return sum(LC)

if 'sequoia' in 'twelve'

42  # of times e is in L
3, 42, 5, 7, 42

the two-terse approaches to those...

remember True == 1
and False == 0
Write each of these functions using list comprehensions...

**def nodds(L):**

```
LC = [for x in L]
return sum(LC)
```

**def lotto_sol(Y, W):**

```
LC = [return sum(LC)
```

**def ndivs(N):**

```
LC = [return sum(LC)
```

**def primesUpTo(P):**

```
return LC
```
hw3pr3: areas from rectangles

Areas of 4 rectangles

Areas of 8 rectangles

y = 2x
Quiz!  
A `range` of list comprehensions...  
Write Python's result for each L.C.:  

```python
[n**2 for n in range(0, 5)]
```

```python
[42 for z in [0, 1, 2]]
```

```python
[z for z in [0, 1, 2]]
```

```python
[s[1::2] for s in ['aces', '451!']]
```

```python
[-7*b for b in range(-6, 6) if abs(b) > 4]
```

```python
[a*(a - 1) for a in range(8) if a % 2 == 1]
```