CS 5: *now recurring…*

Hw3—*due Monday evening—usual time*

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

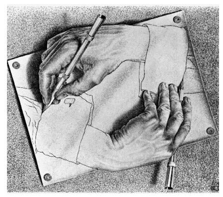
Read Sections 3.6-3.9

We're computationally complete!

What's next?

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

sum range

```
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:])
```

Base Case

Recursive Case

sum $_{\text{list}}$ **range**

list(range(low,hi))

what's cookin' here?

step?

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

excluding **hi**

List Comprehensions

Expression to evaluate for each list element

Name for each list element

The **list** - or **string** to *run*

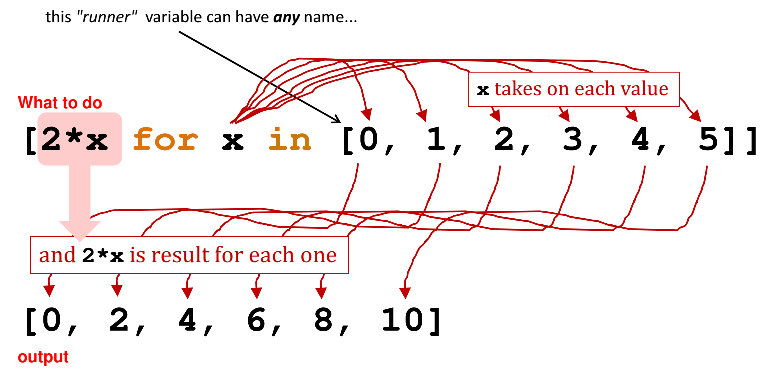
```
[2*x for x in [0, 1, 2, 3, 4, 5]]
```

List Comprehenion

```
[0, 2, 4, 6, 8, 10]
```

result

List Comprehensions



List Comprehensions

```
[10*x for x in [0, 1, 2, 3, 4, 5] if x%2 == 0]
```

result

```
[y*21 for y in list(range(0, 3))]
```

result

```
[s[1] for s in ["hi", "5Cs!"]]
```

result

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!

doubles-counting

```
# 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 list 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))
```

the average #doubles *per 600 rolls*

Designing with LCs

write `[________ for x in list 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

Short and sweet!

```
def fun1(L):          # [7, 8, 9]
    LC = [1 for x in L]
    return sum(LC)
```

```
def letScore(c):
    lots of ifs…
    return score
```

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

# of vowels — 'sequoia' — if

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

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

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

the too-terse approaches to these…

```
LC = [          for c in s ]    top
LC = [          for x in L ]    bottom
```

`remember True == 1`
`and False == 0`

Write each of these functions using list comprehensions...

Go!

Argument: **L**, any list of numbers
Result: the count of odd numbers in **L**
Example: **nodds([3, 4, 5, 7, 42]) == 3**

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

Y are your #s **W** are the winning #s
Arguments: **Y** and **W**, two lists of "lottery" numbers (ints)
Result: the number of matches between **Y** & **W**
Example: **lotto([5, 7, 42, 47], [3, 5, 7, 44, 47]) == 3**

```
def lotto_sol(Y, W):
    LC = [
    return sum(LC)
```

Argument: **N**, an int >= 2
Result: the number of positive divisors of **N**
Example: **numdivs(12) == 6** (1, 2, 3, 4, 6, 12)

```
def ndivs(N):
    LC = [
    return sum(LC)
```

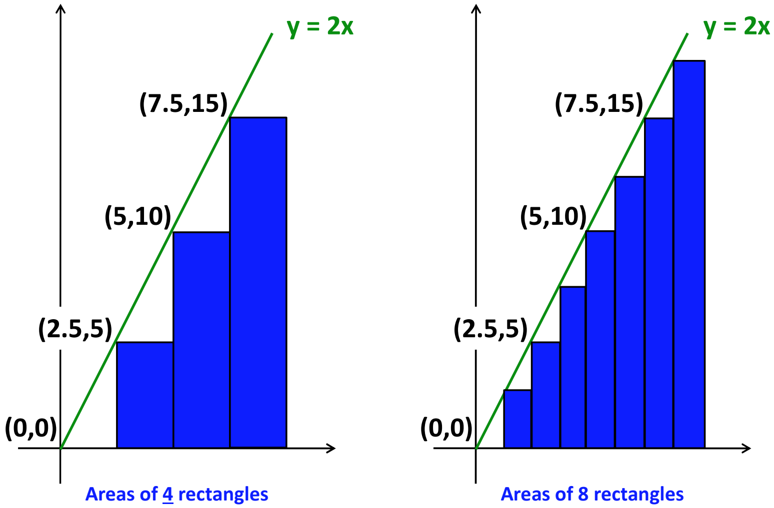
Argument: **P**, an int >= 2
Result: alist of prime numbers up to & incl. **P**
Example: **primesUpTo(12) == [2, 3, 5, 7, 11]**

Extra!

```
def primesUpTo(P):

    return LC
```

hw3pr3: *areas from rectangles*



Quiz!

A **`range`** of list comprehensions...
Write Python's result for each L.C.:

Name(s): ______________________

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

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

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

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

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

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

Got it. But what about that *name*?