A. Finish Recursion

Mathematics

multiplication – like exponentiation (do this an exercise)

log2:

```python
def log2(x):
    # this is the floor log
    count = 0
    while x >= 2:
        x /= 2
        count += 1
    return count
```

gcd:

```python
def gcd(a, b):
    while a != b:
        if a < b: b -= a
        else: a -= b
    return a
```

B. Functional Programming:

- No explicit looping (use recursion instead)
- Functions are first-class objects (unlike C, C++, Java, etc.)
- Variables are only used as placeholders for expressions; assign only once

Recursive functions are almost always true functional programming.

Functional languages:
  - LISP
  - ML
  - Haskell
  - Erlang
  - APL (to some degree)
$$\log_2 (x)$$

```python
count = 0
while x >= 2:
    x /= 2
    count += 1
return count
```

<table>
<thead>
<tr>
<th>$x$</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>
Mixed languages:
most major languages allow a lot of functional programming (e.g. Python)

Functional control structures:
Allow for a type of functionalism, esp. lists
Ruby → Groovy
Python → LC

Map-reduce: do the same computation to each element in a collection (map)
then combine the results of these computations into one thing (reduce)

\[ f = \text{lambda } a, b: \text{a if } (a > b) \text{ else } b \]
\[ f(5, 2) \]
\[ f(2, 5) \]
\[ f(5, 5) \]

```python
from functools import *
reduce(f, [47, 11, 42, 102, 13])
(l lambda a, b: a if (a > b) else b)(5, 2)
```

Use of lambda with recursion:

```python
def filter(somelist):
    if empty(somelist):
        return []
    if head(somelist) < 10:
        return filter(tail(somelist))
    else:
        return insert(head(somelist), filter(tail(somelist)))

def filter(somelist, function):
    if empty(somelist):
        return []
    if function(head(somelist)) == False:
        return filter(tail(somelist), function)
    else:
        return insert(head(somelist), filter(tail(somelist), function))

def empty(L):
    return L == []

def head(L):
    return L[0]

def tail(L):
    return L[1:]

def insert(x, L):
    return [x] + L
```

```
mylist = [5, 10, 11, 7, 12, 15, 99, 2, 8, 88]
print (filter(mylist, (lambda x: x >= 10)))
```

```python
def exactpowerof2(x):
    return 2**math.log(x, 2) == x

exactpowerof2(32) ⇒ True
33 ⇒ False
```

```
2**log2(33)
2 * 15 = 32 ≠ 33
```
```python
def mult(a, b):
    # Add your implementation here

def log2(x):
    if x < 2:
        return 0
    else:
        return 1 + log2(x//2)

# Example usage

# Check if 2^0 equals 1
print(2**0 == 1)
# Check if log2(33) equals some value
print(log2(33))
```

\[
g = \lambda \ a, b: a + b
\]

print( reduce (g, [47, 2, 6, 13, 10]) )

\[
\begin{align*}
49 & \quad 55 \\
68 & \quad 78
\end{align*}
\]

def addup(L):
    if len(L) == 0:
        return 0
    else:
        return L[0] + addup(L[1:])

L = []
head(L)
tail(L)
L[0]
L[1:]
Let's write max using recursion:

def max(L):
    assert L is list, "first arg must be a list"
    if L is empty return a very very very small number (like -99999999999999999999)
    else if first thing in L is smaller than max(tail(L)) return first thing,
    else return max(tail(L))

C. Python comprehensions: a type of map reduce

List and set comprehensions

https://treyhunner.com/2015/12/python-list-comprehensions-now-in-color/

Given a list, let us do so somethings:
  - Filter out some values
  - Make a list from scratch

Simple list comprehensions:

1. Simple list

   mylist = [k for k in range(100)]

2. Even #s

   evens = {x*2 for x in range(0, 100)}
   or
   evens = {x for x in range(0, 100, 2)}

3. Powers of 2
   (do this one in class)

4. List of numbers in a list that are less than a given value (filter problem):

   mylist = [5, 2, 8, 4, 6, 8, 9, 2, 3, 4, 10, 1]
   smalls = {x for x in mylist if x < 6}

5. List of random numbers

   import random
   rands = [random.randint(1, 100) for x in range(10)]
   x = sorted(rands)

6. All the duplicated numbers

   dupes = {x for x in mylist if mylist.count(x) > 1}
   But this causes duplicated numbers to be themselves duplicated! So use a set comprehension:

   {x for x in mylist if mylist.count(x) > 1}
   dupes = set(dupes)

   dupes = {2, 4, 8}
   ᵂзамените
6. All the duplicated numbers

\[ \{x \text{ for } x \text{ in mylist if mylist.count}(x) > 1\} \]

But this causes duplicated numbers to be themselves duplicated! So use a set comprehension:

\[ \{x \text{ for } x \text{ in mylist if mylist.count}(x) > 1\} \]

What if you wanted a sorted list of these?

\[ \text{sorted(list(} \{x \text{ for } x \text{ in mylist if mylist.count}(x) > 1\})\} \]

Interesting side:

\[
\text{if len(x) == len(set(x)):}
\text{print ("All values are unique")}
\text{else:}
\text{print ("Some values were repeated")}
\]

7. Prime #s

```python
def isprime(n):
    return len([x for x in range(2, n) if n%x == 0]) == 0
primes = [x for x in range(2, 10000) if isprime(x)]
```

Twin primes: 2 primes separated by 2

3, 5
5, 7
11, 13
17, 19

How could we write this using list comprehension?

```python
twinprimes = [x for x in range(2, 10000) if isprime(x) and isprime(x+2)]
```

This only gives the first of the pair.

```python
twinprimes = [(x, x+2) for x in range(2, 10000) if isprime(x) and isprime(x+2)]
```

Some tasks need functools or recursion:

- Find the max
- Sum up the list
- See if a target is in the list

Finding the max is kind of complicated and daunting using set and list comprehension!