Built in list functions

x in l

l.index(v) — leftmost position of v in l,
error if v not in l

Updating a list

l = l[1:7] + . . .
l = [8] + 7

} Create a new list object

To update in place

l.append(v) ≡ l+[v]

l.extend([v_1,v_2]) ≡ l+[v_1,v_2]

l.insert(p,x) Insert x **before** position p

l= [3,4,6]
l.insert(2,5) → [3,4,5,6]
l.insert(0,v) [v]+l

l.sort() Sorts l in place, ascending

Removing values from a list

By position del(l[i])

del is a general way to "forget" a value

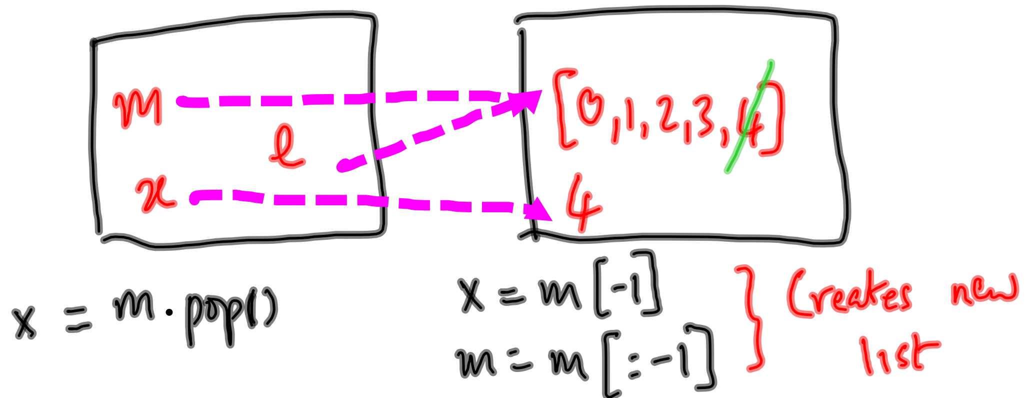
By value l.remove(x) - removes all x

Error if no x in l

del(l[l.index(x)]) - deletes leftmost x

l.pop() removes last element & returns the value

x = l.pop()



Recall

```
def gcd(m,n):
    if m % n == 0:
        return(n)
    else:
        return (gcd(n, m%n))
```

Base Case

Inductive step
Reduce to a smaller problem

```
def factorial(n):

    if n == 0:
        return (1)

    else:
        ans = n * factorial(n-1)
        return (ans)
```

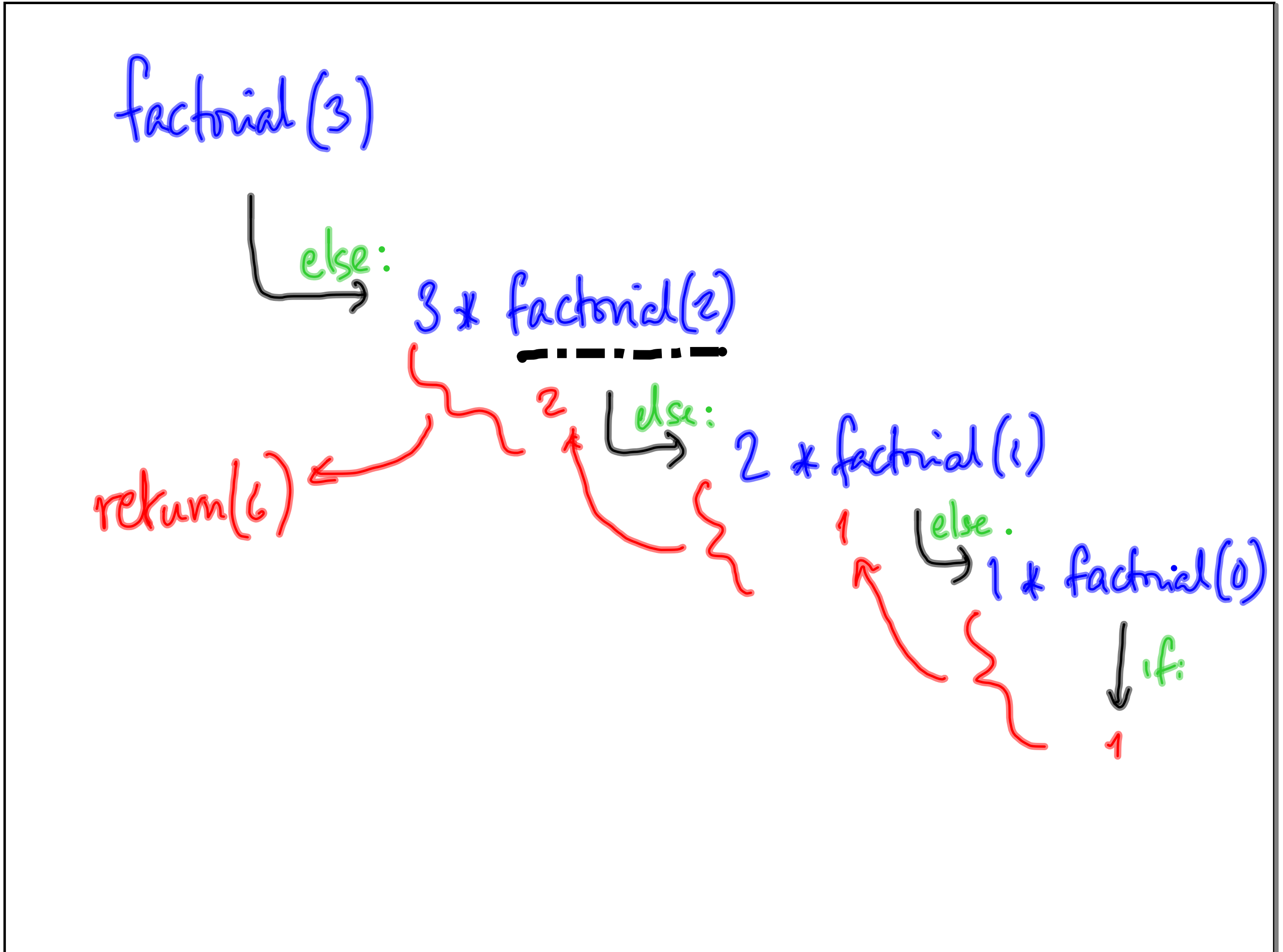
Base Case (annotating `if n == 0: return (1)`)

Inductive step (annotating `ans = n * factorial(n-1)` and `return (ans)`)

The last two lines can be replaced by: `return(n * factorial(n-1))`

$0! = 1$

$n! = n \cdot (n-1)!$


factorial(3)
else:
3 * factorial(2)
2
else:
2 * factorial(1)
return(6)
1
else.
1 * factorial(0)
if:
1

factorial(-1)?

└ -1 * factorial(-2)

└ -2 * factorial(-3)

Infinite descent

"Fix" base case:

```
if n <= 0:
    return(1)
```

What is

$-6 \bmod -4$?

$m \bmod n = r$

$m = qn + r$ +ve

$-6 = +2 \cdot (-4) + 2$ But not in Python!

$\gcd(-6, -4)$

↓

$\gcd(-4, 2)$

```
def fib(n):
  if n==0:
    return(0)
  elif n==1:
    return(1)
  else:
    return(fib(n-2) + fib(n-1))
```

0 1 1 2 3 5 ...

0 1 2 3 ..

2 Base Cases

Use more than 1 smaller value

Arithmetic is defined inductively

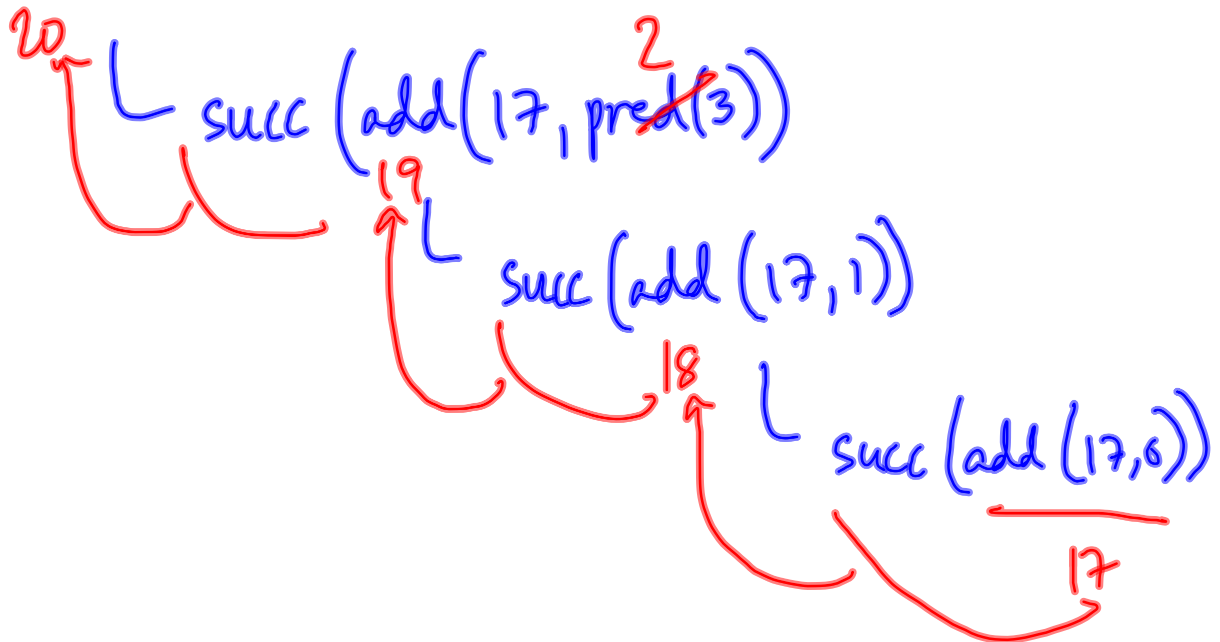
$\mathbb{N}_0 = \{0, 1, 2, \ldots\}$

Assume only a successor function

succ(0) = 1, succ(1) = 2, ...
pred(0) = ?, pred(1) = 0, pred(2) ...

Define addition

```
add(m,n) : if n==0:
               return(m)
           else
               return(succ(add(m, pred(n))))
```


add (17,3)
20
succ (add(17, pred(3)))
2
19
succ (add (17, 1))
18
succ (add (17,0))
17

return (add (succ(m), pred(n)))

17 , 3

18 , 2

19 , 1

20 , 0

20

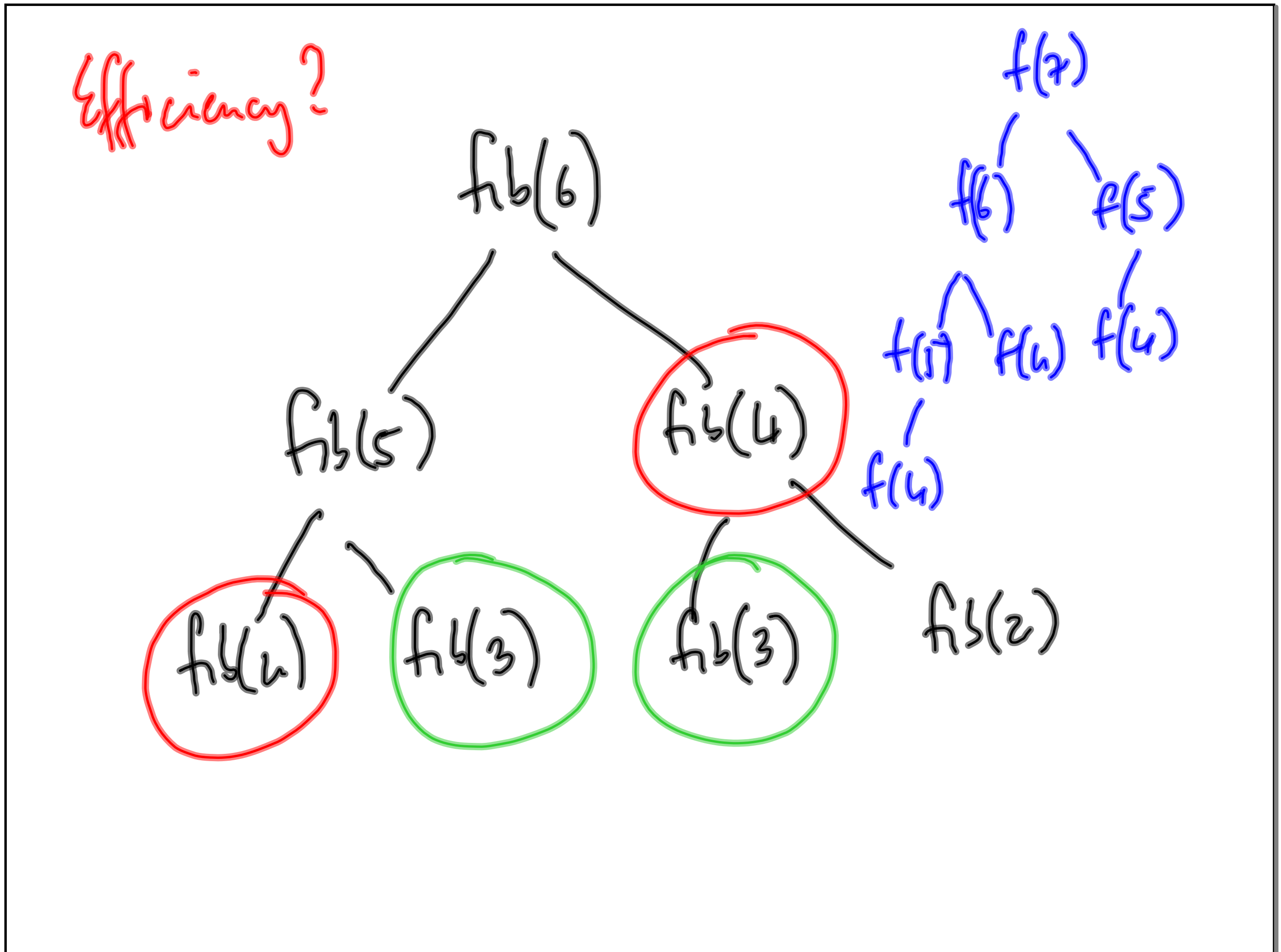
mult (m, n)

m^n

```
if n==0:
    return(0)
else:
    return(add(m, mult(m, pred(n))))
```

power (m, n)

```
if n==0:
    return(1)
else:
    return(mult(m, power(m, pred(n))))
```


Efficiency?
fib(6)
fib(5)
fib(4)
fib(2)
fib(3)
fib(3)
fib(2)
f(7)
f(6)
f(5)
f(5)
f(4)
f(4)
f(4)

Other values also admit inductively defined functions

In particular, lists

Base Case : Empty list

Inductive Case: Strip off one value

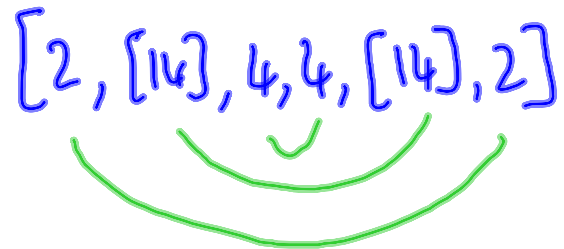
```
def length(l):
    if l == []:
        return(0)
    else:
        return (1 + length(l[1:]))
```

strip off first element

```
def sum(l):
    if l == []:
        return(0)
    else
        return( l[0] + sum(l[1:]))
```

Add up values in l

Check if a list is a palindrome



Check first == last

What remains is a palindrome

```
def palindrome(l):
    if l == []:
        return(True)
    else:
        return( l[0] == l[-1] and
                palindrome(l[1:-1]))
```

l == l.reverse()

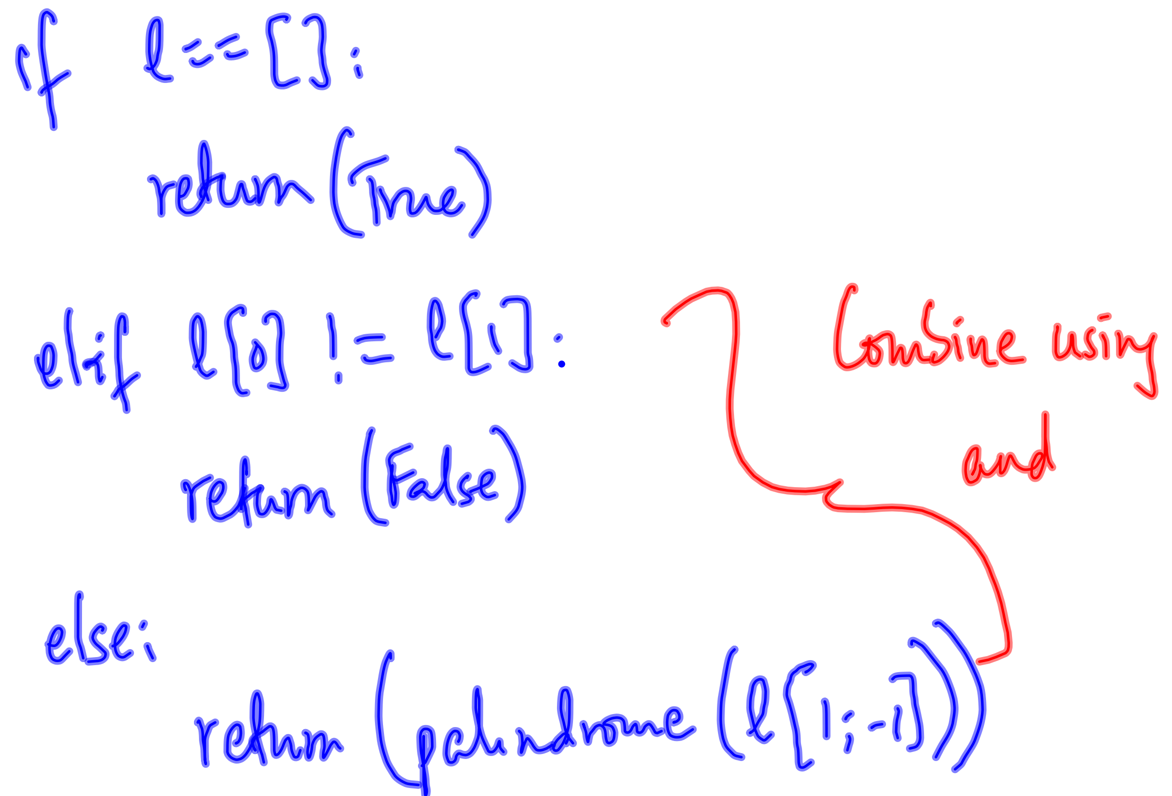
palindrome([6])

l[0] = l[-1] = 6

l[1:-1] = []

↓

True


if l==[]:
return(True)
elif l[0] != l[1]:
return(False)
else:
return(palindrome(l[1:-1]))
Combine using
and