

Function definitions

```
def f(a,b,c,d):
```

```
    |||
```

Call as

$f(x, y, z, w)$

Some flexibility

Call arguments "by name"

$f(a=7, b=x, c=22, d=19)$

OR  $f(b=x, c=22, a=7, d=19)$

} Order  
does  
not  
matter

Can mix args by position & name

$f(3, 7, d=4, c=6)$

... but positions go first

Implicit/default values

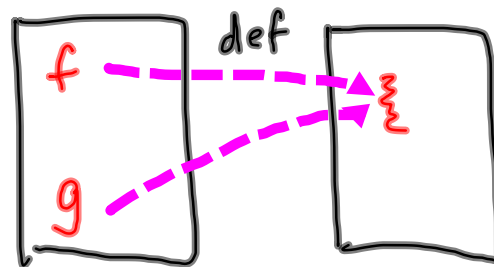
$\text{def } f(x, y=22):$  provide default value  
optional

$f(7):$   $x \leftarrow 7$   $y \leftarrow 22$

$f(7, 9):$   $x \leftarrow 7$   $y \leftarrow 9$

What about:

```
def f(a,b):
    ~~~~~ } value
```



names  
 $g = f$

$g$  &  $f$  point to same function

function "body" is an immutable value

Change  $f$ ,  $g$  remains old function

In Python

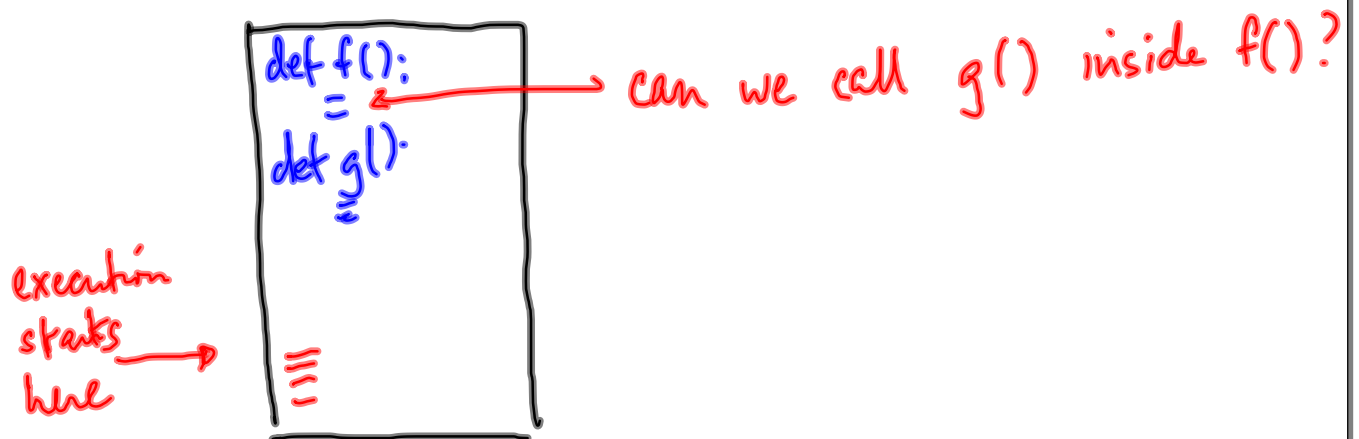
A function definition "assigns" a value to  
a name

Executed like any other stmt

```
if x:  
    def f(--):  
        ≡  
else:  
    def f(--):  
        ≡
```

Technically

like any value, to use a function,  
it must have been defined



$$[0,3] * 2 \rightsquigarrow [0,3] + [0,3] \rightsquigarrow [0,3]$$

What if I want to multiply each value by 2?

$$[0,3] \rightsquigarrow [0,6]$$

for  $i$  in range(0, len(l)):

$$l[i] = 2 * l[i]$$

Suppose I want to square each value?

```
for i in range(0, len(l)):  
    l[i] = l[i] * l[i]
```

In general

```
for i in range(0, len(l)):  
    l[i] = f(l[i])
```

$[x_0, x_1, \dots, x_{n-1}]$



$[f(x_0), f(x_1), \dots, f(x_{n-1})]$

Function called "map"

$\text{map}(f, l)$  Applies  $f$  to each position  
in  $l$

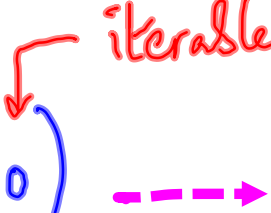
Output is not quite a list

Like output of  $\text{range}(-, -)$

Use  $\text{list}(\text{map}(f, l))$  to get the list



In general

$\text{map}(f, o)$     
iterable object, need not be a list  
produces iterable object

Related function

Extract elements with some property

e.g. Get all even numbers in a list

Use `list(..)` to  
explicitly convert to  
list, when needed

[ 2, 33, 6, 4, 21 ]  
✓ x ✓ ✓ x

```
def iseven(x):
```

```
    return(x%2 == 0)
```

iseven()

[ 2, 6, 4, ]

In place:

```
for i in l:  
    if not (f(l[i])):  
        del (l[i])
```

Do not  
change the  
iterator  
in the  
loop!

filtering a list:

$p: x \mapsto \{\text{true}, \text{false}\}$

$\text{filter}(p, l)$  returns the values in  
 $l$  that  $p$  evaluates to true

$\text{filter}(\text{iseven}, [3, 6, 9, 12]) \rightsquigarrow [6, 12]$

like map, iterable objects etc

Typically map & filter go together

Square all the even numbers in a list

Assume  $\text{sq}(n) \sim n^2$

$\text{iseven}(n) \sim \{\text{True}, \text{False}\}$

$\text{map}(\text{sq}, \text{filter}(\text{iseven}, l))$

$\text{filter}(p, o)$  — iterable object, need not  
be a list

Note:

first argument to `map()` & `filter()`  
is itself a function

"higher order functions"

Another scenario where higher order functions help

mergesort (l)

↳ comparison operation on l  
is implicitly determined

More generally

mergesort (l, cmp)

↳ function to compare pairs  
of values in l

Combining map & filter

$$X = \{x_1, x_2, \dots, x_n\}$$

$$\{x^2 \mid x \in X, \text{even}(x)\}$$

map                      filter

"Set comprehension"

List comprehension

$[f(x) \text{ for } x \text{ in } l \text{ if } p(x)]$

Same as

$list(\text{map}(f, \text{filter}(p, l)))$

Which is clearer to read?



Create a list of 10 0's

$l = [0] * 10$

$l = []$

for  $i$  in range(0,10):

$l.append(0)$

Using list comprehension

$l = [0 \text{ for } i \text{ in range}(0,10)]$

$$[[0, \dots, 0], [0, \dots, 0] \dots [0, \dots, 0]]$$
$$l = \left[ [0 \text{ for } i \text{ in range}(0, 10)] \text{ for } j \text{ in range}(0, 10) \right]$$

Creates a separate list  $[0, \dots, 0]$   
for each  $l[i]$ ,  $i \in \{0, \dots, 9\}$