

Problem to solve

- Given four numbers, days, hours, minutes, and seconds, compute the total number of seconds that is equivalent to them.
- For example, 0 days, 0 hours, 2 minutes, 3 seconds would give 123 seconds. Or 3600 seconds would equal to 0 days, 1 hours, 0 minutes, and 0 seconds.

How to tackle the problem?

- We can use a list to represent the days, hours, minutes, and seconds that are fed to the function.
- The function then should return as the result of computation the total number of seconds.
- We'd like the program run something like

```
>>> convert_to_seconds([0, 0, 2, 3])
123
>>> convert_to_seconds([0, 1, 0, 0])
3600
```

How functions *look*...

```
def convert_to_seconds(in_list):
    """ convert_to_seconds(in_list): Converts a
        list of [days, hours, minutes, seconds]
        into seconds
        Input: a list of four int's
    """
    seconds = in_list[0] # number of seconds
    minutes = in_list[1] # number of minutes
    hours = in_list[2] # number of hours
    days = in_list[3] # number of days
    return seconds + minutes*60 + hours*3600
    + days*24*3600
```

Annotations in the code block:

- name**: points to `convert_to_seconds`
- input to function (parameters)**: points to `in_list`
- docstring**: points to the triple-quoted string
- code block**: points to the function body
- comments**: points to the inline comments
- return statement**: points to the `return` line

How functions *work*...

Assume you have written three functions:

```
def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1 * x
```

What is `demo(-4)` ?

How functions work...

```
def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1 * x
```

```
demo
x = -4
return -4 + f(-4)
```

>>> `demo(-4)` ?

How functions work...

```
def demo(x):
    return x + f(x)

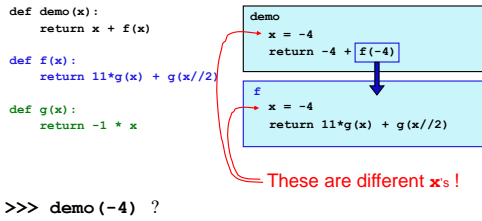
def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1 * x
```

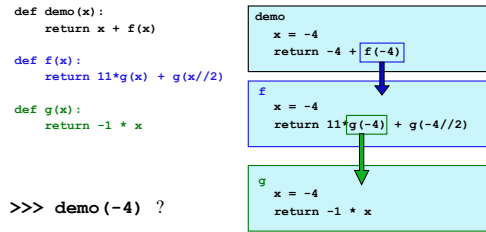
```
demo
x = -4
return -4 + f(-4)
↓
f
x = -4
return 11*g(x) + g(x//2)
```

>>> `demo(-4)` ?

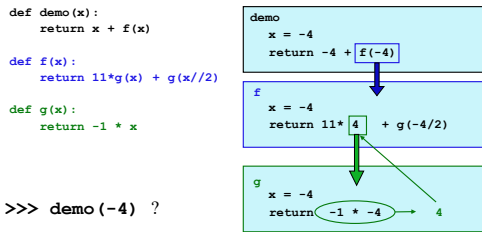
How functions work...



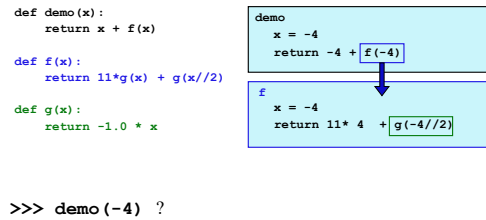
How functions work...



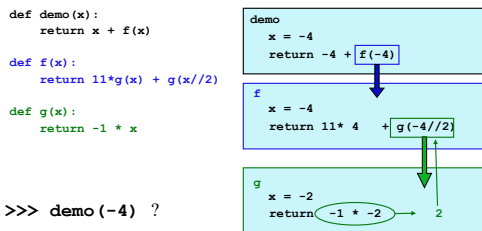
How functions work...



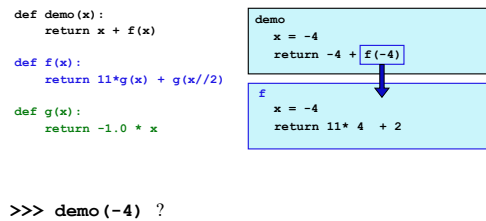
How functions work...



How functions work...



How functions work...

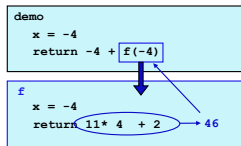


How functions work...

```
def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1.0 * x
```



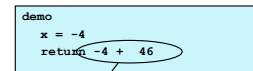
```
>>> demo(-4) ?
```

How functions work...

```
def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1.0 * x
```



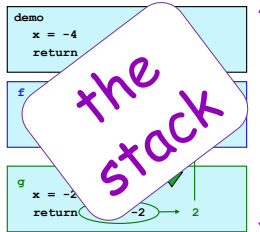
```
>>> demo(-4) → 42
42
```

Function *stacking*

```
def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1.0 * x
```



(1) keeps separate variables for each function call...

(2) remembers where to send results back...

return != print

```
def dbl(x):
    """ doubles x """
    return 2*x

>>> answer = dbl(21)
>>> answer
42
```

```
def dblPR(x):
    """ doubles x """
    print (2*x)

>>> answer = dblPR(21)
42
>>> answer
>>>
```

print just prints! It does not return anything

- **Print is for people**

return provides the function call's *value* ...