Introduction to Python for Political Scientists

Week 2

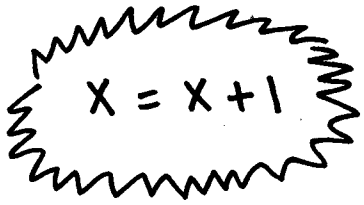
Functions()
[Lists]
(Tuples)

review

indentation is important

```
x = 0
while True:
    ..... do something
    ..... till user gets fed up/
    ..... computer melts/
          the universe ends
```

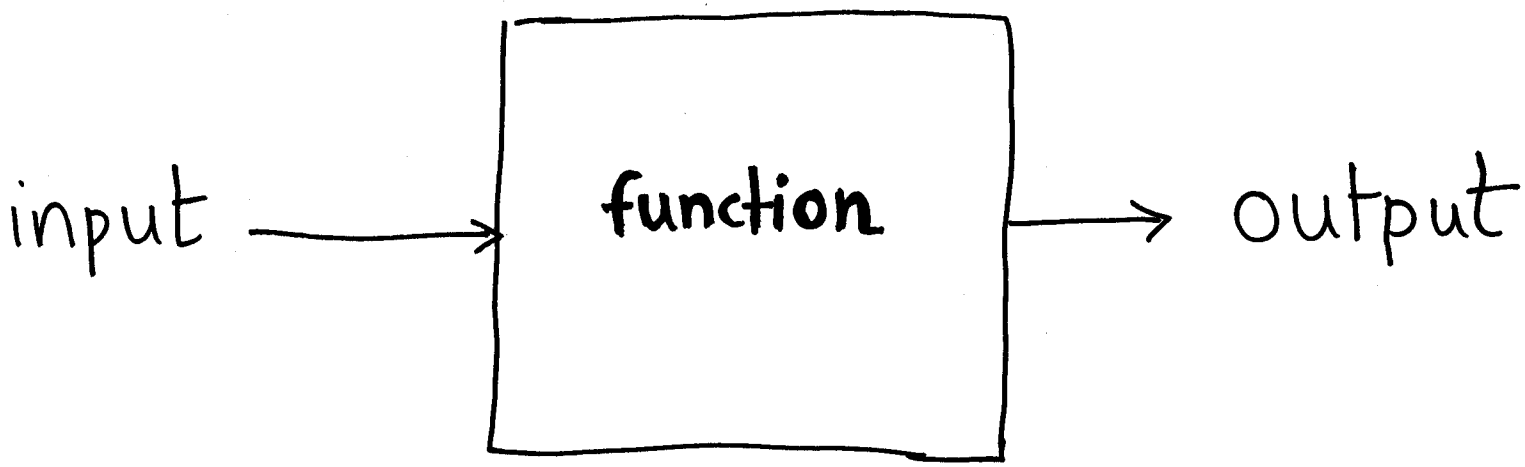
while loop



NEVER gets executed !!

loops (while/for)
functions
if/else
...

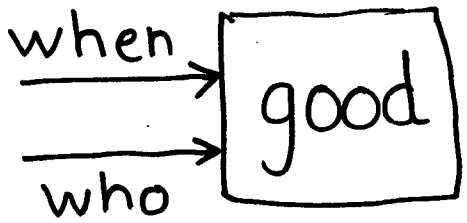
boxes of statements



function keyword

parameters

```
def good(when, who):
    print("Good " + when + " " + who)
```

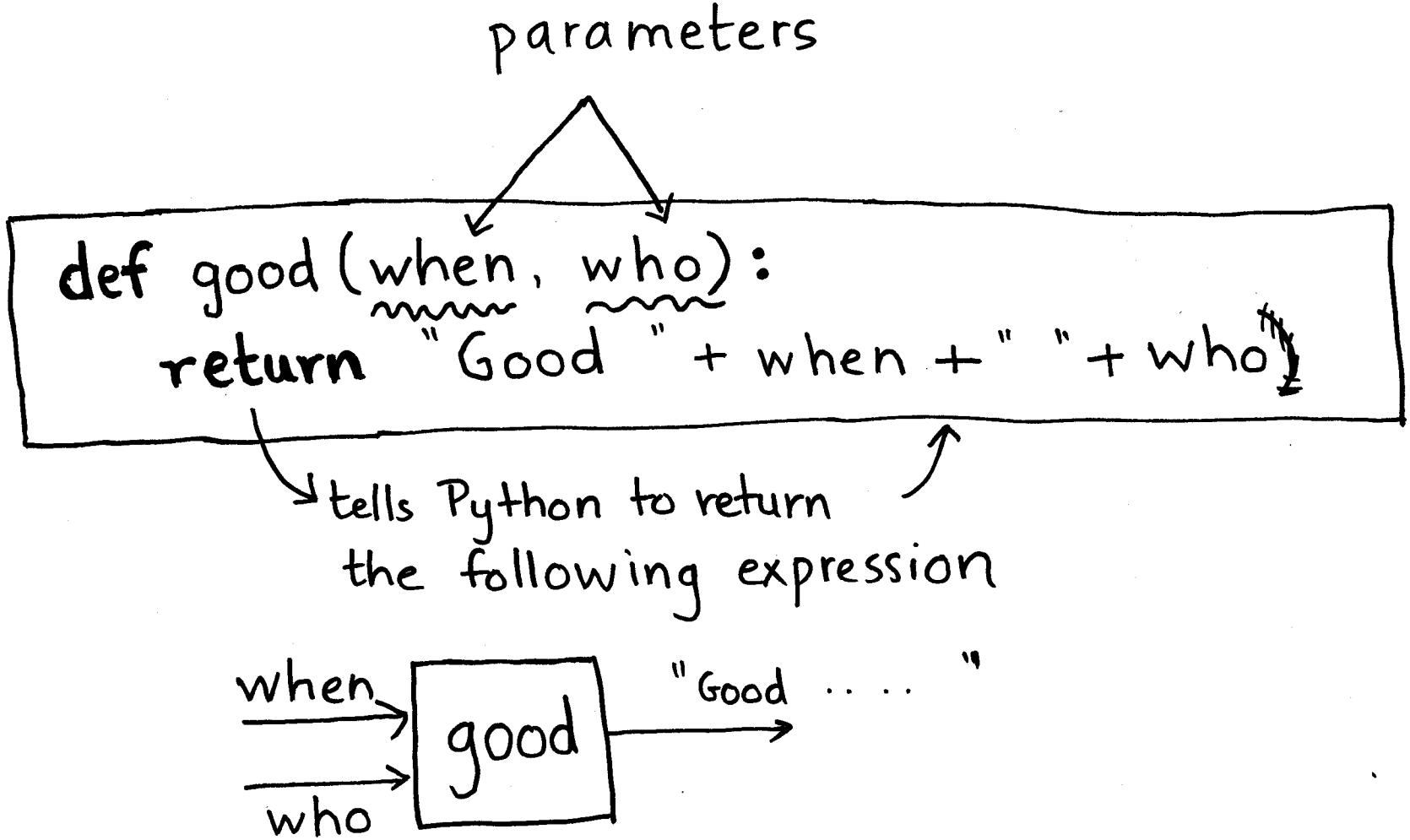


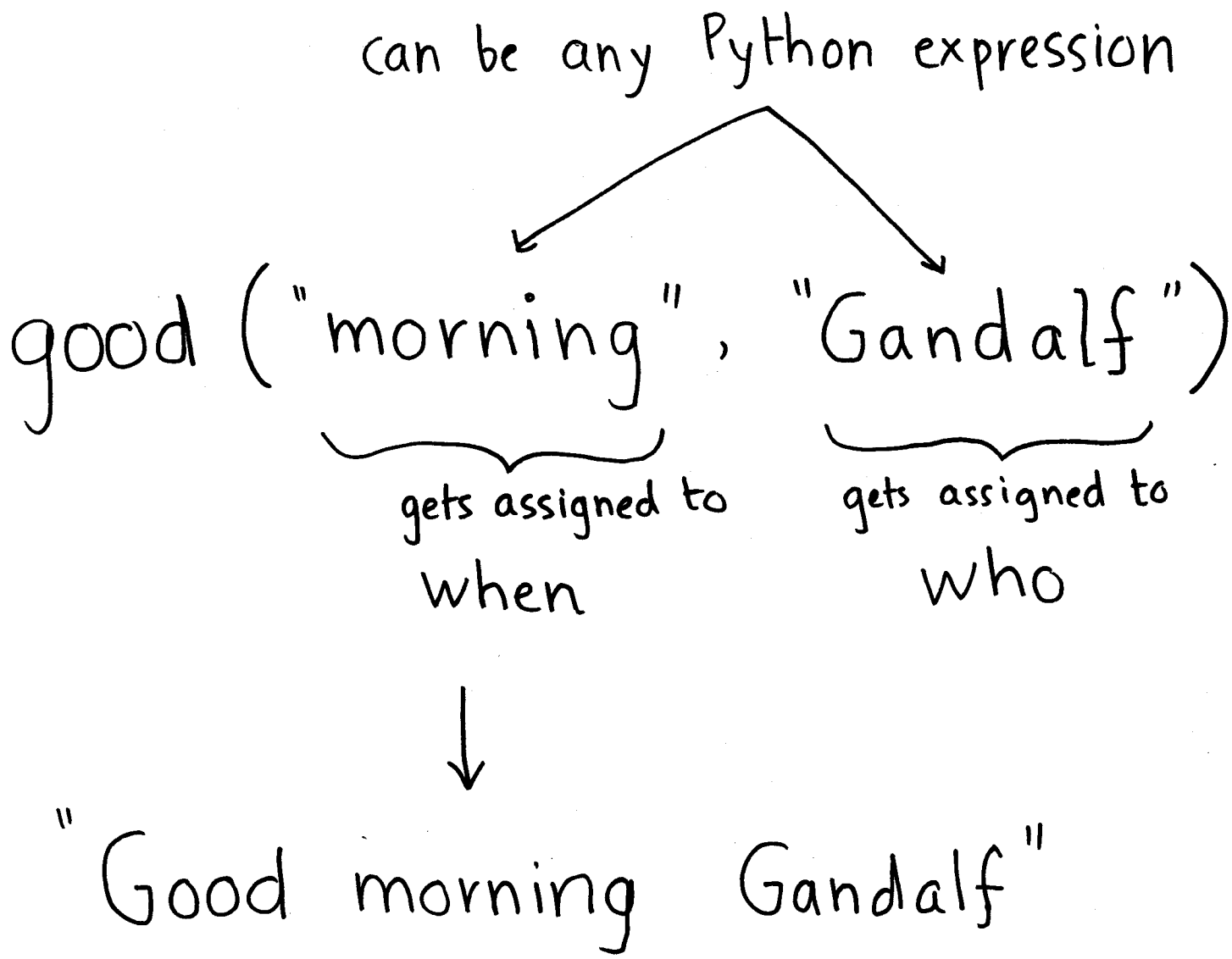
?

It's printing the greeting.

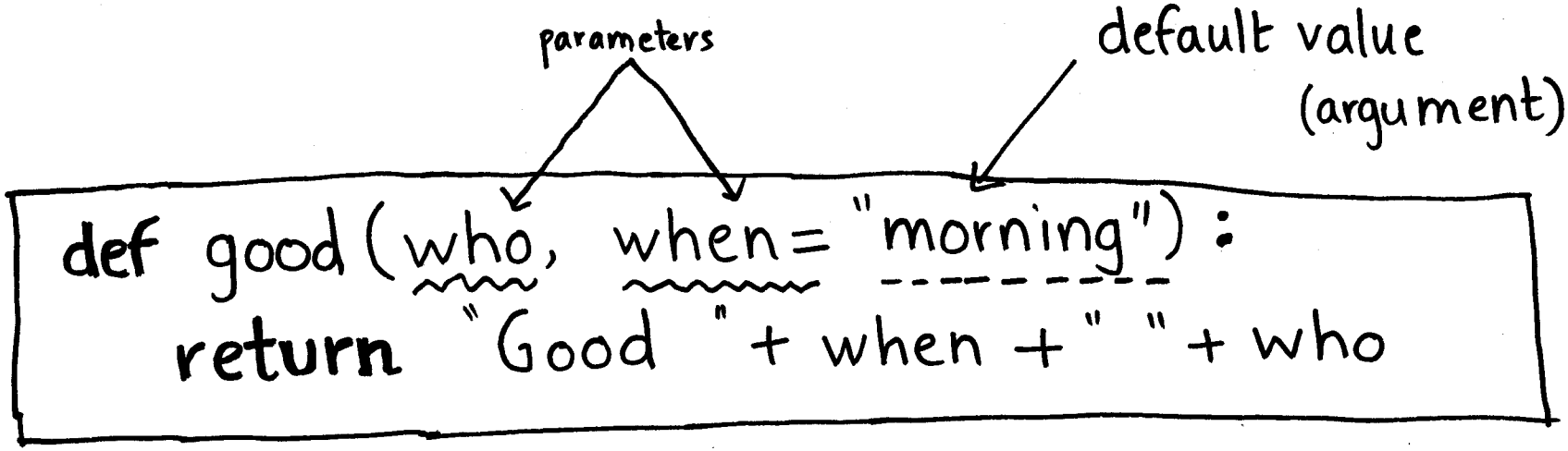
Is that the output?

Nope →


parameters
def good(when, who):
return "Good " + when + " " + who
tells Python to return
the following expression
when
who
good
"Good . . . "


can be any Python expression
good ("morning", "Gandalf")
gets assigned to
when
gets assigned to
who
"Good morning Gandalf"

You are usually saying "Good morning..."



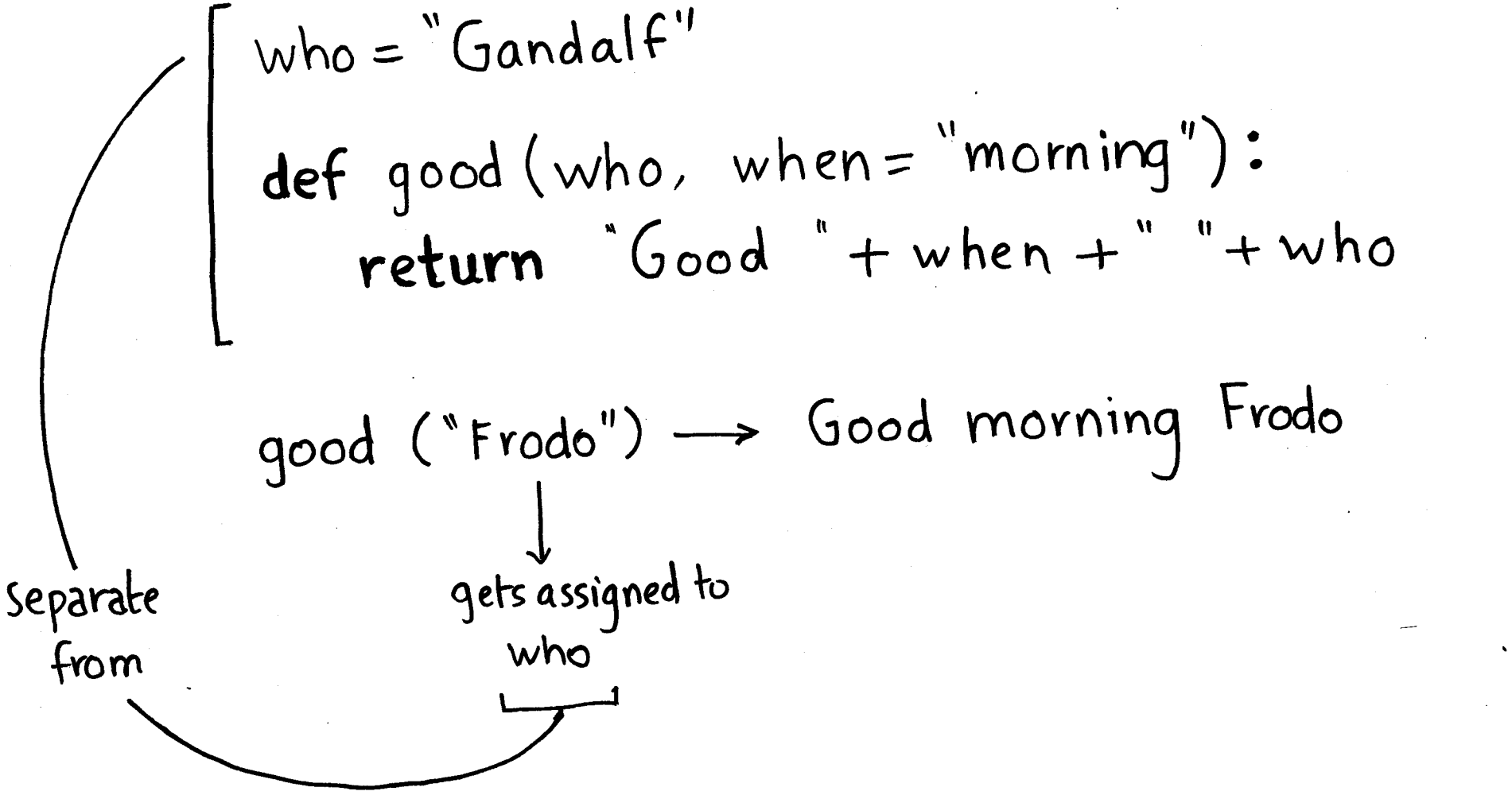
good("Gandalf") → "Good morning Gandalf"

good("Gandalf", "evening") → "Good evening Gandalf"

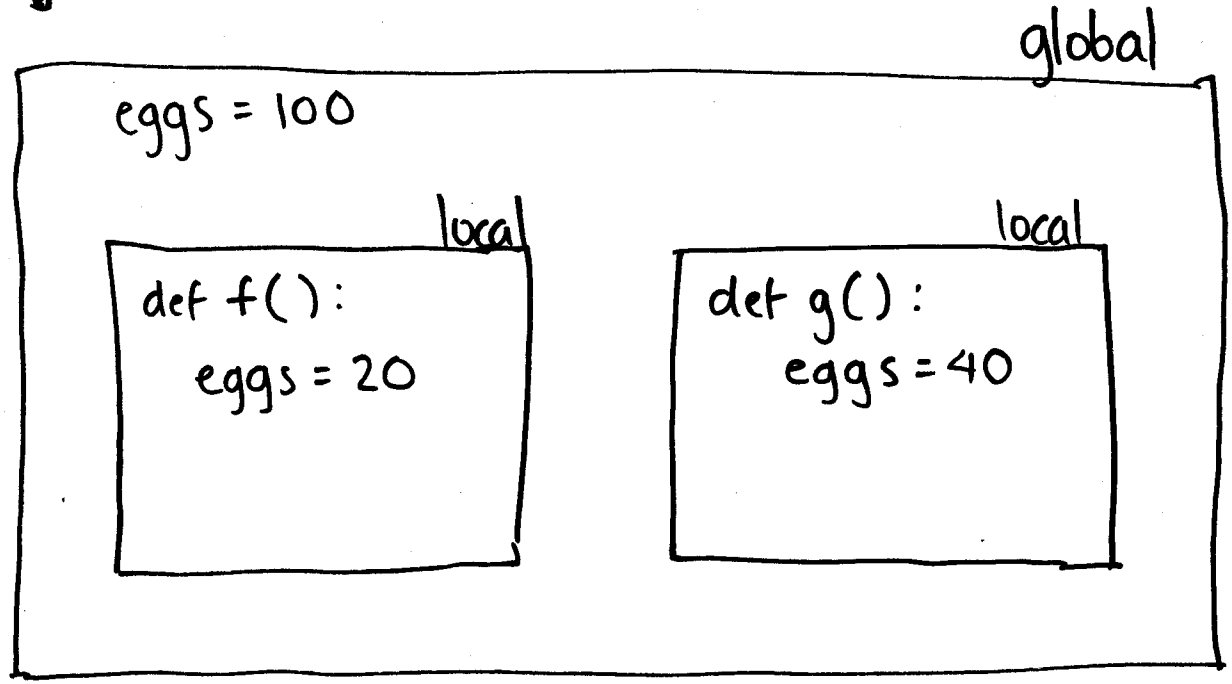
good(when="bye", who="Gandalf") → "Good bye Gandalf"

using named parameters allows us to change the order.

Often used for optional arguments → ~~another~~ name for values passed to the function


who = "Gandalf"
def good(who, when="morning"):
return "Good " + when + " " + who
good("Frodo") → Good morning Frodo
gets assigned to who
Separate from

scope



All these eggs are different!

scope

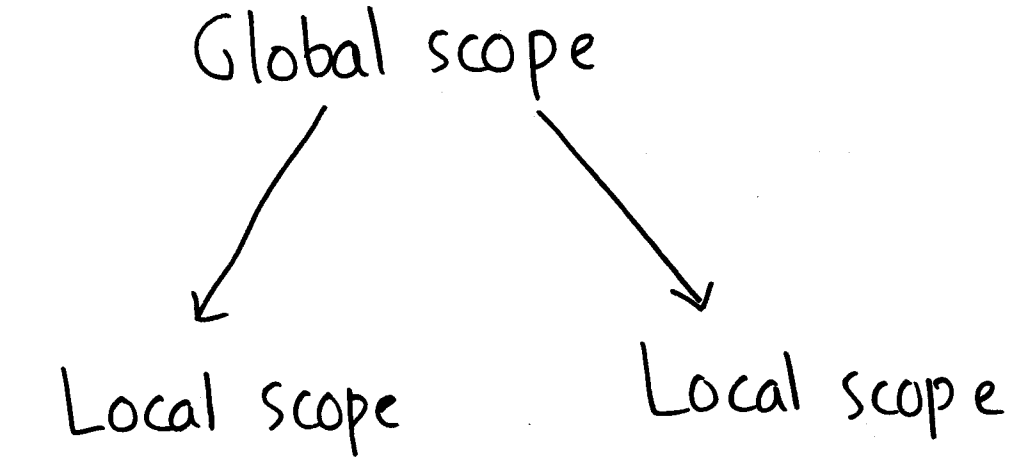
One can see into the box one is contained in.

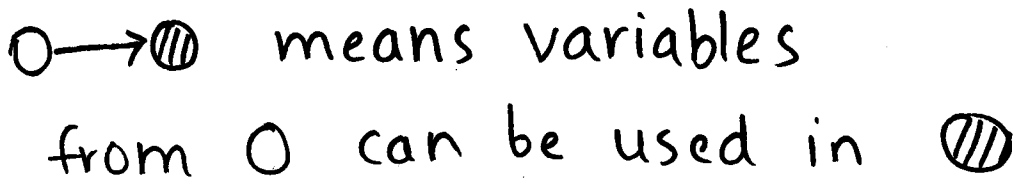
unless....

```
def f():
    global eggs
    eggs = 40
```

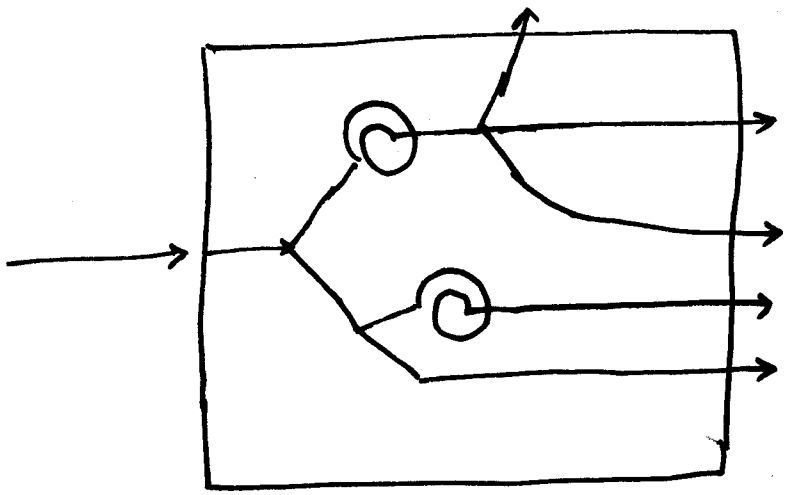
Local scopes do not share variables.

Variables in local scope override those in global scope.


Global scope
Local scope
Local scope


means variables
from O can be used in

You can **return** from multiple places as your code can have multiple paths (if/else, loops)



Like NULL, null, nil, etc. in other languages

with the caps

None.

is the default **return** value.

```
>>> x = print("hello world")
>>> print(x)
None
```


Lists.

```
>>> days = ['Sunday', 'Monday', 'Tuesday',
            'Wednesday', 'Thursday',
            'Friday', 'Saturday']

>>> len(days)
7
```

for loops

```
for i in range(n):
    ____
    ____
    ____
```

i runs through the **iterator** (list-like) expression, taking values sequentially in the loop.

range(n) ⟶ 0 ... n-1

default is 0

range (start, stop) ⟶ 0 ... start .. (stop - 1)

range (start, stop, step) ⟶ start, start + step,
start + 2 * step,
....(upto) stop - 1.

can be negative

range (5, 10) ⟶ 5, 6, 7, 8, 9

range (2, 10, 2) ⟶ 2, 4, 6, 8

range (10, 5, -2) ⟶ 10, 8, 6

Need not all be the same type

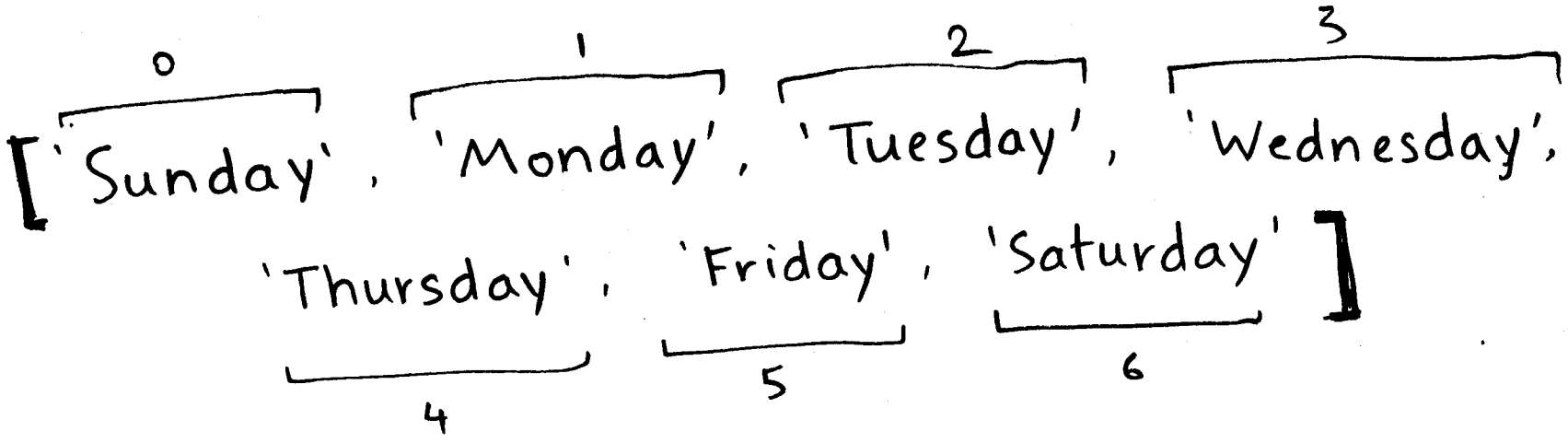
[1, 'Sunday', 0, True, 5.0]

Generally, they are, like days.

[ ] indexing

↓ individual

accessing elements from the list



The first element is indexed 0.

access: days [index]

days[0] → 'Sunday' ...

[ ] slicing (L is the list)

L[m:n] ⟶ new list from m^{th} to n-1th element

L[m:] ⟶ new list from m^{th} to end

L[:m] ⟶ new list upto m-1th element

↓ can be negative, then it is offset from the end

L[-1] ⟶ last element

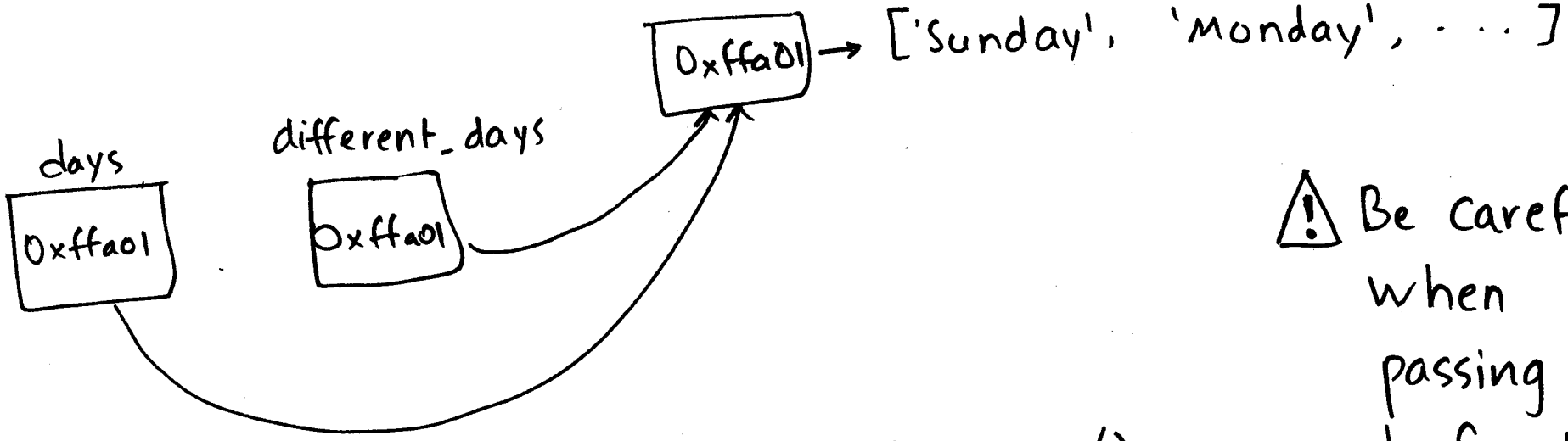
L[:-1] ⟶ upto, but excluding last element

appending and deleting and concatenating

```
>>> days
['Sunday'. . . . .  ']
APPEND >>> days.append ['Freeday')
>>> len(days)
8
>>> days
DELETION >>> del days[-1]
CONCAT >>> days = days + ['Freeday', 'Fryday', 'Sleepday']
ASSIGN/ >>> days[0] = 'Zeroday'
MUTATE
```

Lists are stored as references

```
>>> different_days = days
>>> different_days.append("Notanotherday")
>>> different_days == days
True
```



⚠ Be careful when passing lists to functions!

```
>>> different_days = days.copy()
```

('a', 1, True)

Strings are like tuples of characters

Tuples are like lists but they are immutable

s = ('h', 'e', 'l', 'l', 'o')

s[:-1] = 'hell'

return x, y → tuple

SWAP (multiple assignments)

a, b = b, a

>>>

Sieve of Eratosthenes

week-2/sieve.py

sieve(30)

2 3 4 5 6 7

8 9 10 11 12 13

14 15 16 17 18 19

20 21 22 23 24 25

26 27 28 29 30

sieve(30)

2 3 ~~4~~ 5 ~~6~~ 7

~~8~~ 9 ~~10~~ 11 ~~12~~ 13

~~14~~ 15 ~~16~~ 17 ~~18~~ 19

~~20~~ 21 ~~22~~ 23 ~~24~~ 25

~~26~~ 27 ~~28~~ 29 ~~30~~

sieve(30)

2 3 ~~4~~ 5 ~~6~~ 7

~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13

~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19

~~20~~ ~~21~~ ~~22~~ 23 ~~24~~ 25

~~26~~ ~~27~~ ~~28~~ 29 ~~30~~

sieve(30)

2 3 ~~4~~ 5 ~~6~~ 7

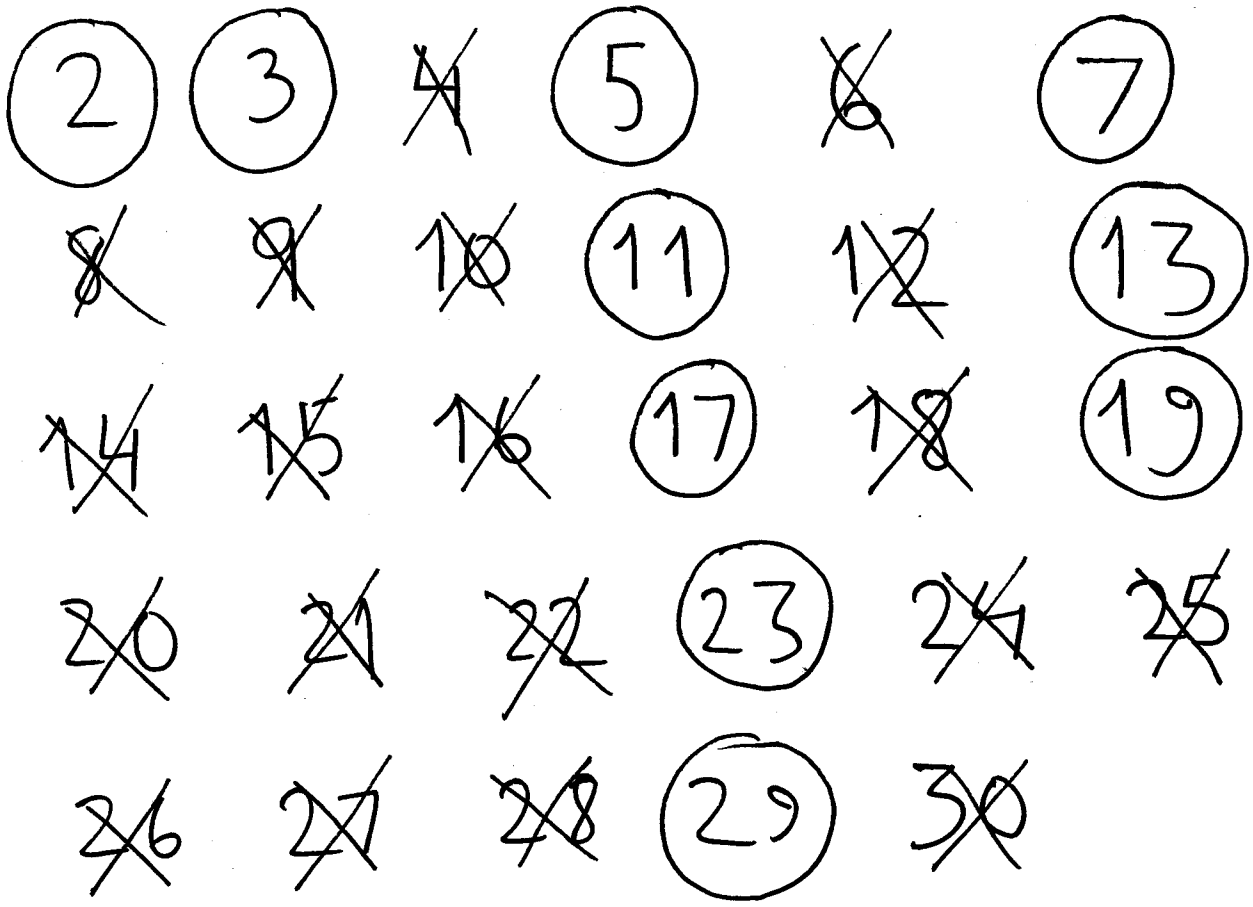
~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13

~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19

~~20~~ ~~21~~ ~~22~~ 23 ~~24~~ ~~25~~

~~26~~ ~~27~~ ~~28~~ 29 ~~30~~

sieve(30)


2 3 4 5 6 7
8 9 10 11 12 13
14 15 16 17 18 19
20 21 22 23 24 25
26 27 28 29 30