Names and values

Scalar int, float, bool, no char

↓

✓ a and b a or b LOGICAL

✗ a & b a | b BITWISE

Lists – example of "sequence"

Lists

Strings

Tuples

List written as [1, 3, [2, 3.5], False]

l = [1, 3, [False]] l[2][0] ⇝ False

l[i] is value at position i, 0 ≤ i < length

Concatenation is + (++ in Haskell)

l = [1,3] + [[False]] ⇝ [1, 3, [False]]

l[i:j] "slice" [l[i], l[i+1], .., l[j-1]]

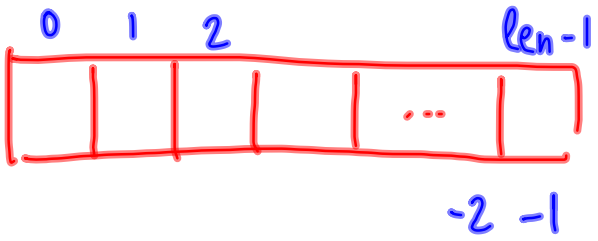
Prefix l[0:j], also l[:j]

len(l) is length of l

Suffix l[i:len(l)], also l[i:]

"Full" slice : l[:]

Negative indices are interpreted from right end



String: h = "hello"

x = '"world"'

y = '''"John's ball"'''

} Single or double quote

Triple quote!

Like lists: h = "hello" x = '"world"'

h[2] ⇝ "l"

h + x ⇝ 'hello"world"'

h[2:4] ⇝ "ll"

Tuples

pair = (3,4)

info = ("AB", 62, 3.5, False)

What about updates?

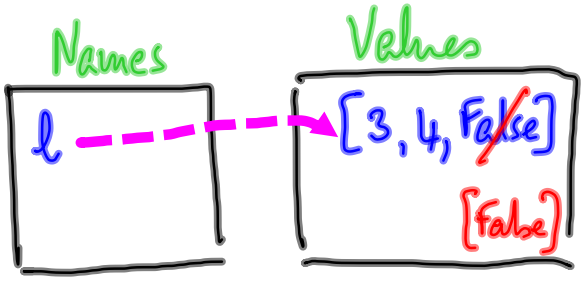
l = [3, 4, False]

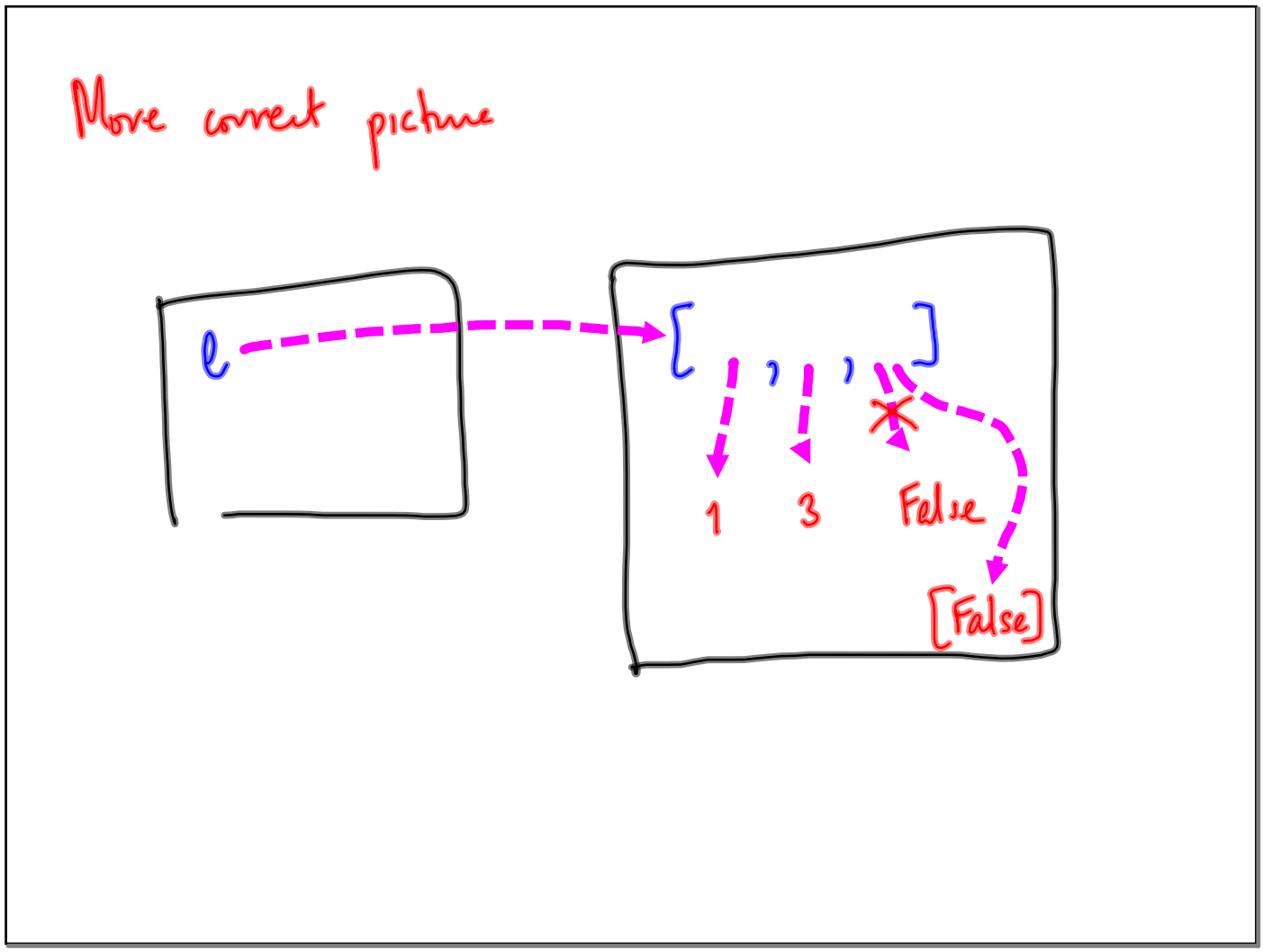
Want to change False to [False]

Update in place

l[2] = [False]

What does this mean?




More correct picture
e
[, ,]
1
3
False
[False]

h = "hello" Want "help!"

h[3] = "p", h[4] = "!" X

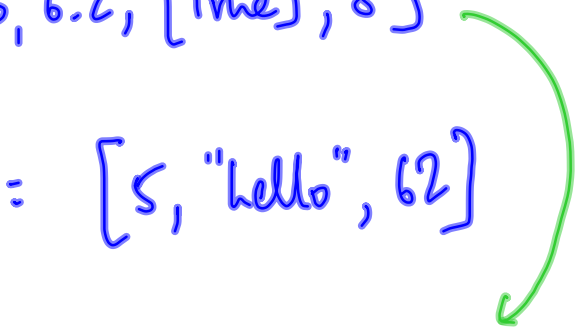
Lists are mutable

Strings & tuples (and scalars) are immutable

Instead: h = h[0:2] + "p!"

In a list, can directly update a slice

```
l = [1, 3, 6.2, [True], 8]
l[1:4] = [5, "hello", 62]
```



```
[1, 5, "hello", 62, 8]
l[1:3] = [16, 18, 19]
[1, 16, 18, 19, 62, 8]
```

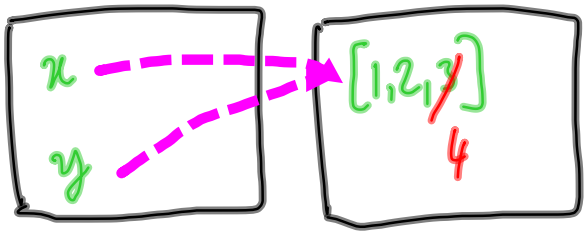
Can contract a slice

`l[i:j] = []` deletes the slice

`x = [1,2,3]`

`y = x` DOES NOT COPY THE LIST

`y[2] = 4`



`y = x[:]` slice always makes a new object

Use full slice to make a copy of a list

Function definitions in Python:

```
def examplefn(x):
    y = 2*x
    return(y)
```

indent uniformly

Modifying control flow

```
if conditional-expression :
    <suite>
else :                      OPTIONAL
    <suite>
```

```
def f(x):
    if x%2 == 0:
        y = 2*x
    else:
        y = 3*x
    return(y)
```

Or

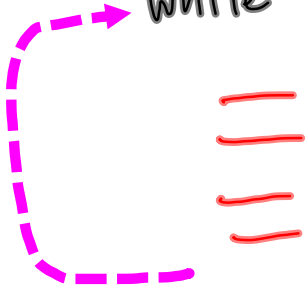
```
def f(x):
    if x%2 == 0:
        y = 2*x
        return(y)
    else:
        y = 3*x
        return(y)
```

collapse return (2*x)

← fn exits

What happens if we omit this?

Do something repeatedly




```
def mylog(x,n):
    ans = 0
    while x >= n :
        x = x//n
        ans = ans + 1
    return(ans)
```