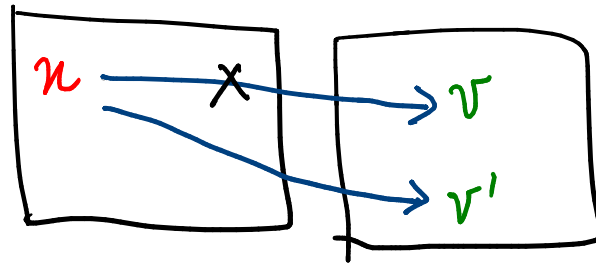


# Python Names & Values

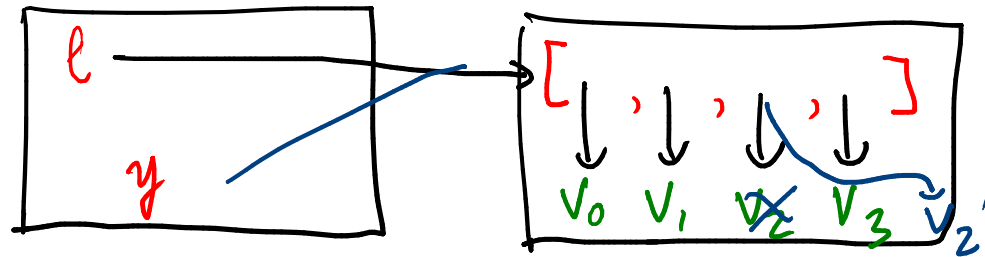
Immutable Integer, Float, Boolean



Immutable update  
creates a new link

$x = v'$

Mutable "In Place"



$l[2] = v_2'$  Updates  $y[2]$  as well

Strings are like lists, but immutable.

"hell"  $\leadsto$  "help"  
 $\left. \begin{array}{l} y = \text{"hell"} \\ y[3] = \text{"p"} \end{array} \right] \times$

Slice - segment of a list

$l[i:j]$   $[l[i], l[i+1], \dots, l[j-1]]$

Works for strings as well

$l = [0, 1, 2, 3]$

$l[2] = 2$

$l[2:3] = [2]$

$s = \text{"hell"}$

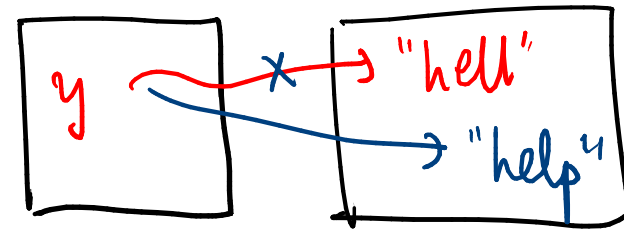
$s[2] = \text{"l"}$

$s[2:3] = \text{"l"}$

Recall that  $+$  is concatenation

Want  $y = \text{"hell"}$  to become  $\text{"help"}$

$y = y[0:3] + \text{"p"}$



Useful shortcuts

$l[:j] \equiv l[0:j]$

$l[i:] \equiv l[i:\text{len}(l)]$

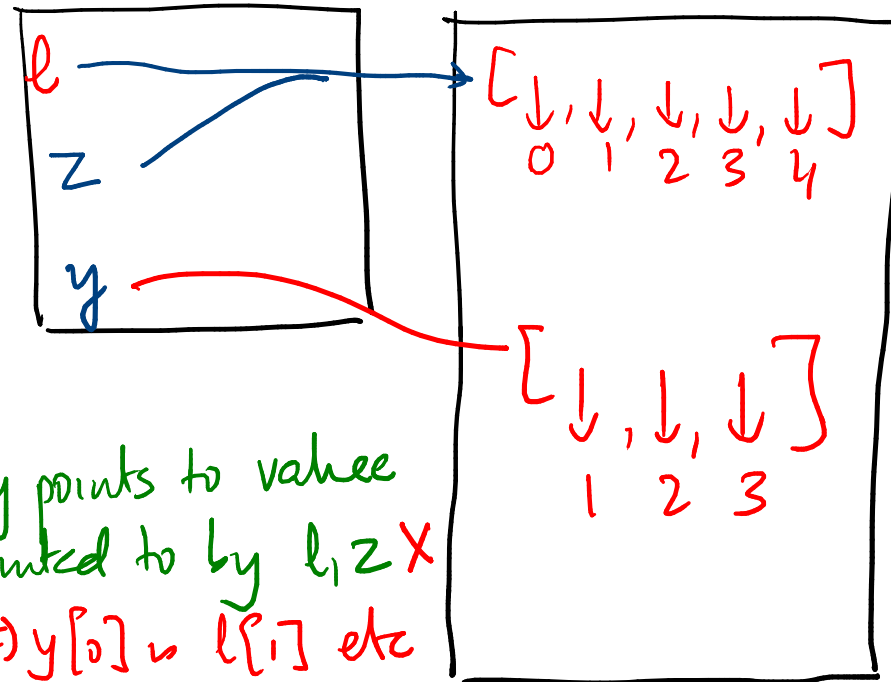
$l[:]$  - Copy of  $l$

this is why  
everything in  
Python goes to  
 $n-1$

$l = [0, 1, 2, 3, 4]$

$z = l$

Updates to  $z$  affects  
 $l$  & vice versa



$y = l[1:4]$

directly points to value  
pointed to by  $l, z$  X  
 $\Rightarrow y[0] = l[1]$  etc

copies, fresh value

To safely copy a list

$z = l[:]$  = "full slice"

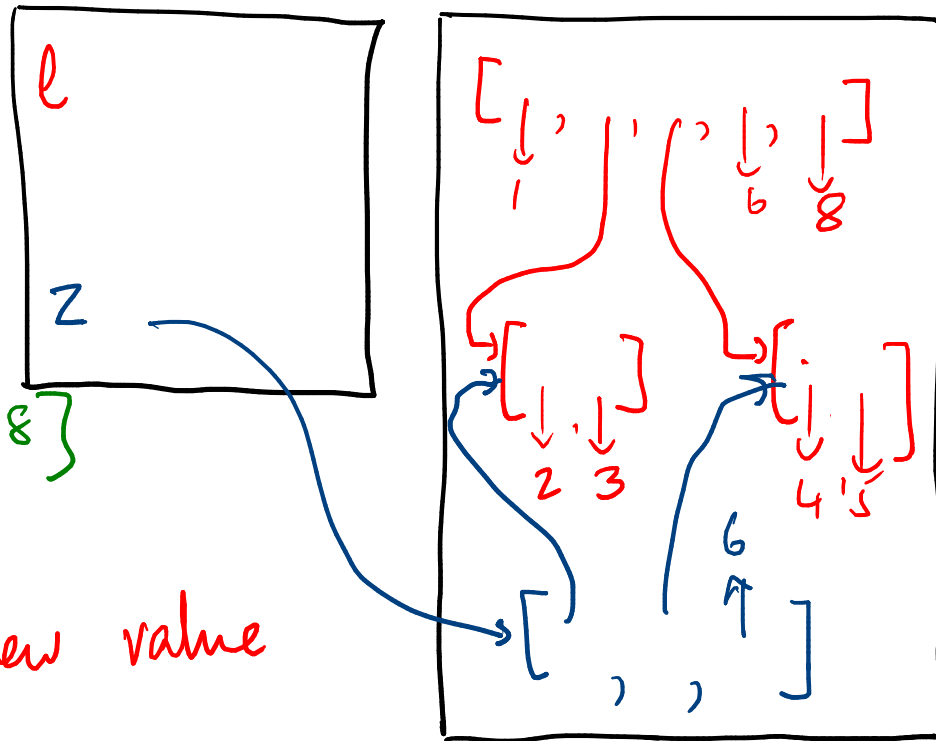
$$l = [1, [2, 3], [4, 5], 6, 8]$$

$$z = l[1:4]$$

$$l = [1, [2, 3], [4, 5], 6, 8]$$

New Value

Reassigning creates new value



Assign to a slice

$l = [0, 1, 2, 3, 4]$

$l[1:4] = [7, 8, 9]$

In place update

$l[1:4] = [9, 10, 11, 12]$

"expands"

$l[1:4] = [9]$

$l[1:4] = []$

deletes slice

$[0, 9, 10, 11, 12, 4]$   
 $[0, 9, 4]$

Check Python docs for functions on strings, lists

`len(l)`

`l.append(v) ↔ l = l + [v]`

`l.extend(l')` ↔ `l = l + l'`

in place

new value



Control flow

Procedural / imperative program

Sequence of instructions

Basic instruction: assignment  
name = value

Conditional instructions

Repetition

if (condition) :  
     s1  
     s2  
     s3  
     :  
     sk  
 else:  
     s'  
     :  
     s'e

optional  
 expression that evaluates to True/False

Shortcut    0 is False  
               "" is False  
               [] is False  
               None - "non value"  
 Everything else is true

3 way branch

if c1 ~  
c2 ~  
c3 ~

if c1 :  
=

else :

if c2 :  
=

else :

if c3 :

else :

if c1:

elif c2:

elif c3:

else: Optional

## Repetition:

- Know the number of repetitions in advance
- Repeat until some condition holds

### Type 1

for name in list:

≡

for  $x$  in  $[v_1, v_2, \dots, v_k]$

$\left( \begin{array}{l} \downarrow x = v_1 \\ \downarrow x = v_2 \dots \end{array} \right) x = v_k$


### Type 2


while C:

≡

← assumed that something happens to change C

Define units of "work" - functions

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

  $a=1$   
 $b=2$   
 $c=3$   
 $d=1$

$f(1, 2, 3, 4)$

```
def gcd(m,n):
```

```
    val = min(m,n)
```

```
    while ( val > 0 ):
```

```
        if ( m%val == 0 ) and ( n%val == 0 ):
```

```
            return (val)
```

```
        [ else:
```

```
            val = val - 1 ]
```

```
    val = val - 1
```

```
X = X + V
X = X - V
X = X * V
```

```
X += V
X *= V
```

Write your function in a file `mygcd.py`

load it manually into Python interpreter

```
from mygcd import *
```

Repetition

for i from 1 to n

range(i, j) sequence  $[i, i+1, \dots, j-1]$   
like a list

range(i, j, k)  $i, i+k, i+2k, \dots$

range(j)  $0, \dots, j-1$

can be negative



Converting types

`print(x)` implicitly converts `x` to String

`str(x)` converts `x` to string

`int(v)` converts to integer

`list(x)`