Lists in Python

What is a List?

- An ordered set of values:
  - Ordered: 1st, 2nd, 3rd, …
  - Values: can be anything, integers, strings, other lists
- List values are called *elements*.
- A *string* is an ordered set of characters so it is “like” a list but not exactly the same thing.

Create a new List:

- Assignment:

x = [1,5,'egg',[2,3]]

A couple of integers, a string and a "nested" list, all in the same list.

- Calling the *range()* function:

```
y = range(5)
print y
[1, 2, 3, 4]
```

range(a) **returns all the integers from *0* to *a-1***

```
y = range(1,5)
print y
[1, 2, 3, 4]
```

range(a,b) **returns all the integers from *a* to *b*, including *a*, excluding *b.***

```
y = range(1,15,2)
print y
[1, 2, 3, 4]
```

range(a,b,c) **returns all the integers *a, a+c, a+2c, …up* to *b*, including *a*, excluding *b.***

The Empty List

```
x = [ ]
```

- The empty list is usually used to initialize a list variable but not give it any useful elements.

Accessing Elements:

- List elements are accessed via integer indexes starting at 0 and working up.

```
numbers = [ 3, 87, 43]
print numbers[1], numbers[2], numbers[0]
87 43 3

x = 3
print numbers[x-2]
87

Print numbers[1.0]
TypeError: sequence index must be integer

print numbers[3]
TypeError: list index out of range

print numbers[-1]    # a negative index counts back
3                    # from the end of the list
                     # index -1 is the last element

print numbers[-3]
TypeError: list index out of range
```

Accessing Many Elements:

- By index value, one at a time (called *list traversal*)

```
# list of a known size
horsemen = [‘war’, ‘famine’, ‘pestilence’, ‘death’]
i = 0
while i < 4:
    print horsemen[i]
    i = i + 1

# or if you don’t know how long the list is
i = 0
length = len(horsemen)
while i < length:
    print horsemen[i]
    i = i + 1

war
famine
pestilence
death
```

always safer to use *len* as an upper bound

always I < length; never I <= length

List Membership:

- You simply ask if a value is *“in”* or *“not in”* a list.
- This is always a *True/False* question.

```
horsemen = [‘war’, ‘famine’, ‘pestilence’, ‘death’]
if ‘debauchery’ in horseman:
    print ‘There are more than 4 horsemen of the apocolipse.

print ‘debauchery’ not in horsemen
1
```

Loop Operator for Lists

- We have already seen that a *while-loop* can be used to “traverse” a list.
- There is also a special *for-loop* notation.

```
horsemen = [‘war’, ‘famine’, ‘pestilence’, ‘death’]
for horseman in horsemen:
    print horseman
```

a different variable

- Exercise: Print out a list of ten 0s and 1s.

```
for x in range(20):
    print x%2
```

List Operations:

- Add two lists:

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
print c
[1, 2, 3, 4 ,5, 6]
```

- Repeat a list many times:

```
a = [1, 2, 3]
print a*3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

- Exercise: Create a list of 20 zeros.

```
zeros = [0]*20
```

List Slices:

- Sometimes you just want a sub-list (*slice*) of a list.

list[a:b] means

list[a], list[a+1], …, list[b-1]

 # all list elements with indexes from a to b;
 # including a and excluding b

```
vowels = ['a', 'e', 'i', 'o', 'u']
print vowels[2:4]
['i', 'o']

# how do you print out the last element?
print vowels[2:]
['i', 'o', 'u']
```

- Exercise: What does *vowels[:3]* mean?

['a', 'e', 'i']

Exercise:

- What does *[:]* mean?

```
>>> print vowels[:]
[‘a’, ‘e’, ‘i’, ‘o’, ‘u’]
```

Lists are *Mutable* (Their values can change):

```
fruit = [‘apple’, ‘orange’, ‘pear’]
fruit[1] = ‘fig’
print fruit
[‘apple’, ‘fig’, ‘pear’]
```

- However it gets trickier when you try to add something to a list and not just replace something . . .

List Slices Used to Modify a List:

- Suppose you are keeping an ordered list:

```
names = [‘adam’, ‘carol’, ‘henry’, ‘margot’, ‘phil’]
```

- And you want to add *kate*. Assignment doesn’t work!

```
names = [‘adam’, ‘carol’, ‘henry’, ‘margot’, ‘phil’]
names[2] = ‘kate’

print names
[‘adam’, ‘carol’, ‘kate’, ‘margot’, ‘phil’]
```

- You can add an element by squeezing it into an empty slice between two list elements:

```
names = [‘adam’, ‘carol’, ‘henry’, ‘margot’, ‘phil’]
names[2:2] = ‘kate’

print names
[‘adam’, ‘carol’, ‘henry’,  ‘kate’, ‘margot’, ‘phil’]
```

Starting at index 2 but not including 2; ie, empty

List Deletion:

- Using the *del* operator

```
names = [‘adam’, ‘carol’, ‘henry’, ‘margot’, ‘phil’]
del names[3]

print names
 [‘adam’, ‘carol’, ‘henry’, ‘phi
```

- Replacing an element with an empty list

```
names = [‘adam’, ‘carol’, ‘henry’, ‘margot’, ‘phil’]
names[3:4] = [ ]

print names
 [‘adam’, ‘carol’, ‘henry’, ‘phil’]
```

- Deleting slices

```
names = [‘adam’, ‘carol’, ‘henry’, ‘margot’, ‘phil’]
del names[1:4]

print names
 [‘adam’, ‘phil’]
```

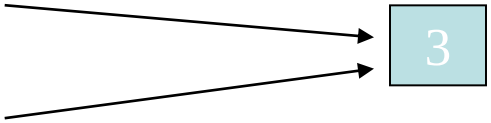
Objects and Values:

- Remember:

```
x = 3
y = 3

print id(x), id(y)
135045528 135045528
```

- So memory looks like:

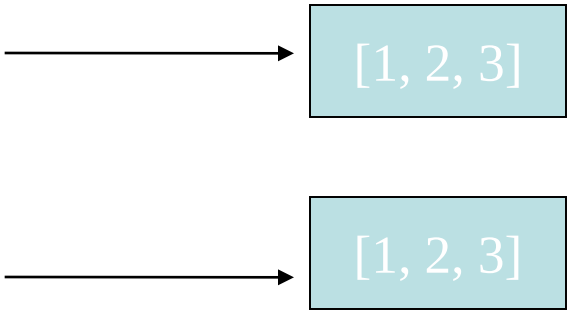


- Two places in memory called *x* and *y*, both pointing to a place with a 3 stored in it.

Lists, Objects and Values

- Lists are different:

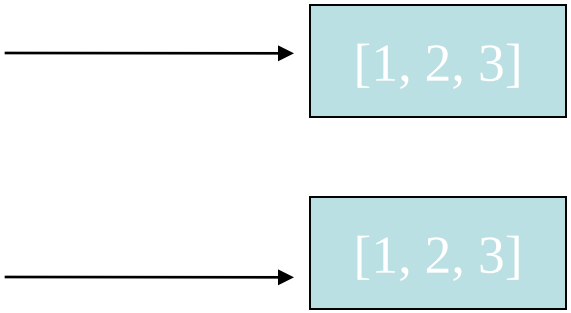
- So this time the memory state picture is:



Lists, Objects and Values

- However, if we use assignment:

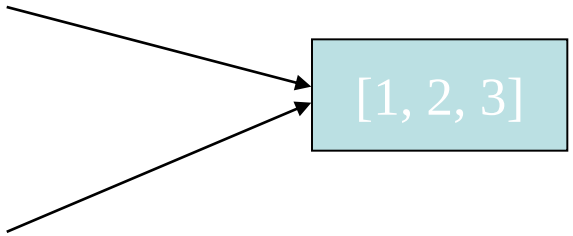
- So this time the memory state picture is:



Aliasing

- However, if we assign one variable to another:

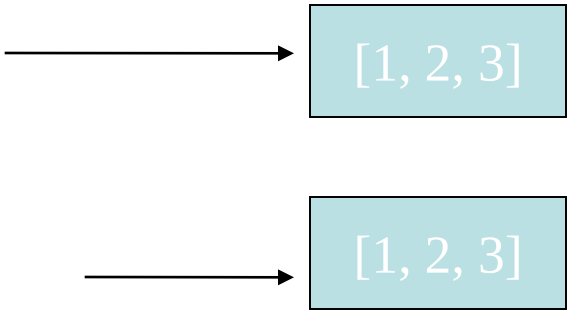
- So this time the memory state picture is:



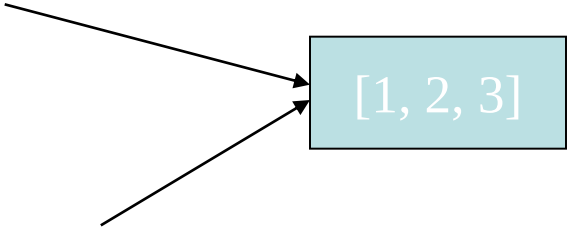
- More importantly, changing *b* also changes *a*

Lists are like:

- Money:



- Or also Credit Cards:



Cloning a List:

- Cloning means making an exact but separate copy:
- Not Cloning:

```
a = [1, 2, 3]
b = a

print id(a), id(b)
135023431 135023431
```

- Cloning:

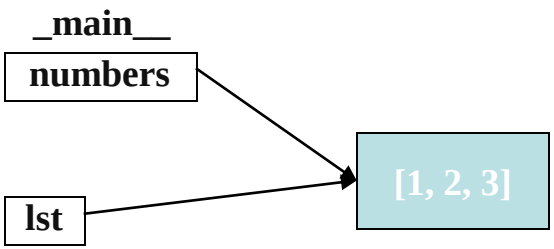
```
a = [1, 2, 3]
b = a[:]            # slices are always separate lists

print id(a), id(b)
135023431 13502652
```

List Parameters:

- We create a function and we pass a list as its argument.

```
def head(lst):        # returns the first element of a list
    return lst[0]      #  list remains unc
```



Changing List Arguments:

```
def tail(lst):
    return lst[1:]

numbers = [1,2,3]
rest = tail(numbers)

print numbers
[1,2,3]

print rest
[2,3]
```

```
def deleteHead(lst):
    del lst[0]

numbers = [1,2,3]
deleteHead(numbers)
print numbers
[2,3]
```

Lists and Strings:

- The String module has very useful list functions. Suppose we have

  ‘The rain in Spain falls mainly in the plane’

- And we want

  [‘The’, ‘rain’, ‘in’, ‘Spain’, ‘falls’, ‘mainly’, ‘in’, ‘the’, ‘plane’]

- There is a string function that will split up a string into a list of *tokens*.

```
str = ‘The rain in Spain falls mainly in the plane’
tokens = string.split(str)
print tokens
 [‘The’, ‘rain’, ‘in’, ‘Spain’, ‘falls’, ‘mainly’, ‘in’, ‘the’, ‘plane’]
```

Lists and Strings:

- The opposite to *string.split()* is *string.join().*

```
tokens = [‘The’, ‘rain’, ‘in’, ‘Spain’, ‘falls’, ‘mainly’, ‘in’, ‘the’,  ‘plane’]
str = string.join(tokens,‘ ‘)
print str
‘The rain in Spain falls mainly in the plane’
```

join( ) **will join tokens with whatever separator you wish to use. In this case the separator is a space ‘ ‘.**