



LISTS

LISTS

- a list is an ordering of items
 - It can be a sequence of strings, numbers and so on...
- Actually you have seen a sequence before in a string. You can think of a string as a particular kind of list. Think about it for a moment...
- We can create a list with the square brackets []

```
1  numbers = [5, 7, 4, 3, -8]
2
3  words = ["movie", "actor", "batman"]
4
5  floats = [3.14, 6.5, 2.3, 76.90]
6
7  empty_list = []
```

LISTS

- When naming variables that store a list it is a good choice to use plural names. That way it is clear that the variable stores more than one value.
- Values of a list are separated by a comma
- An empty list is simply an opening square bracket and a closing square bracket with nothing between them

A list of integers stored in a variable called numbers

A list of strings stored in a variable called words

A list of floats stored in a variable called floats

An empty list stored in a variable called empty_list

```
1 numbers = [5, 7, 4, 3, -8]
2
3 words = ["movie", "actor", "batman"]
4
5 floats = [3.14, 6.5, 2.3, 76.90]
6
7 empty_list = []
```

LISTS

- Lists in Python can store different types within the same list
 - This list contains strings, integers and a float

```
values = ["batman", 7, "actor", 4, 3, -8, 3.14]
```

- Printing lists is pretty easy

```
1 values = ["batman", 7, "actor", 4, 3, -8, 3.14]
2
3 print(values)
```

LISTS

- The for loop can be used with lists

fruit_basket is a list of strings

We decide to call the iteration variable **fruit**

Then we print each value in the list one at a time

```
1 → fruit_basket = ["apple", "banana", "melon"]
2
3  for fruit in fruit_basket:
4  | → print(fruit)
```

LISTS

- Elements of lists are indexed
 - This means that we can access individual elements of a list by using their index
 - But do note that the index is zero based, that means that the first element of a list has the index 0 , not 1

```
fruit_basket = ["apple", "banana", "melon"]
```



LISTS

- We can access individual elements of a list via the square brackets
 - We put the square brackets behind the name of the variable that stores the list and within the brackets we specify the index

This line will print "apple"

This line will print "melon"

```
1 fruit_basket = ["apple", "banana", "melon"]
2
3 print(fruit_basket[0])
4
5 print(fruit_basket[2])
```

LISTS

- Lists are mutable (changeable)
 - This means that you can change the content of a list

Here we are changing the content at index 0 to the string pear

Here we are changing the content at index 2 to the string orange

```
1 fruit_basket = ["apple", "banana", "melon"]
2
3 # this prints ["apple", "banana", "melon"]
4 print(fruit_basket)
5
6 fruit_basket[0] = "pear"
7 fruit_basket[2] = "orange"
8
9 # this prints ["pear", "banana", "orange"]
10 print(fruit_basket)
```


LISTS

- We can use the **in** operator to check whether a given value is in a list
 - The **in** operator evaluates to **True** or **False**

If the string "apple" is in the list
fruit_basket this text will be printed

If the string "apple" is not in the list
fruit_basket this text will be printed

```
1 fruit_basket = ["apple", "banana", "melon"]
2
3 if "apple" in fruit_basket:
4     print("Lets eat!")
5 else:
6     print("Ahhhh, I only eat apples")
7
```

LISTS

- Lists are objects and objects have methods(functions) associated with them
 - You will learn more about objects later
- What this means is that we have some predefined operations we can use on lists such as:
 - Adding to a list
 - Removing an element from a list
 - Sorting a list
 - Etc.

LISTS

Mutable !!!

- Remember, lists are ~~immutable~~! That means we can change
- That also means that some of the methods associated with lists will not create a new list but simply change the list
 - These methods will have the return value of None
 - `.append()`
 - `.extend()`
 - `.pop()`
 - `.insert()`
 - `.remove()`
 - `.sort()`
 - `.reverse()`
- This can be confusing at first, especially because we have seen that many string methods return a new string!
- But remember, lists and string have many things in common but they are not the same thing!

LISTS

- We can use the `append()` method to add an element to the back of a list
- Methods are functions that "belong" to datastructures we are working with
 - The `append` method "belongs" to a list
 - We call methods with the `dot(.)` operator

Here we are adding the string "pineapple" to the list `fruit_basket`

```
1 fruit_basket = ["apple", "banana", "melon"]
2
3 # this prints ["apple", "banana", "melon"]
4 print(fruit_basket)
5
6 fruit_basket.append("pineapple")
7
8 # this prints ["apple", "banana", "melon", "pineapple"]
9 print(fruit_basket)
```

The value that is to be added to the list comes between the parentheses

LISTS

- The `remove()` method removes an element from a list
 - We need to put the value that is to be removed between the parentheses

Here we call the remove method and pass it the string "banana"

As can be seen here the string "banana" has been removed from the list

```
1 fruit_basket = ["apple", "banana", "melon"]
2
3 # this prints ["apple", "banana", "melon"]
4 print(fruit_basket)
5
6 fruit_basket.remove("banana")
7
8 # this prints ["apple", "melon"]
9 print(fruit_basket)
```

LISTS

- The `pop()` method removes an elements from the back of a list
 - The `pop()` method doesn't take any parameters, that is you don't need to put anything between the parentheses

```
1  fruit_basket = ["apple", "banana", "melon"]
2
3  # this prints ["apple", "banana", "melon"]
4  print(fruit_basket)
5
6  fruit_basket.pop()
7
8  # this prints ["apple", "banana"]
9  print(fruit_basket)
```

LISTS

- We can insert values at a given index in a list using the insert method
- The insert method takes two parameters
 - The first one represents the index in the list and the second one the value that should be added at that index

Here we add the string orange at index(position) 1

```
1 fruit_basket = ["apple", "banana", "melon"]
2
3 # this prints ["apple", "banana", "melon"]
4 print(fruit_basket)
5
6 fruit_basket.insert(1, "orange")
7
8 # this prints ["apple", "orange", "banana", "melon"]
9 print(fruit_basket)
```

LISTS

- Here are some functions commonly used with lists

```
1  a_list = [1,2,3,4]
2
3  length = len(a_list)
4  sum_of_list = sum(a_list)
5  max_value = max(a_list)
6  min_value = min(a_list)
```

- `max(some_list)`: returns the largest element, all elements of the list must be the same type
- `min(some_list)`: returns the smallest element, all elements of the list must be the same type
- `sum(lst)`: returns the sum of the elements, all elements must be of a numeric type

LISTS



Remember! Lists are zero indexed.
That means that the first element
is at index 0

LISTS

- Lists work a bit differently than the data types we have been working with so far
 - Especially regarding to the computers memory

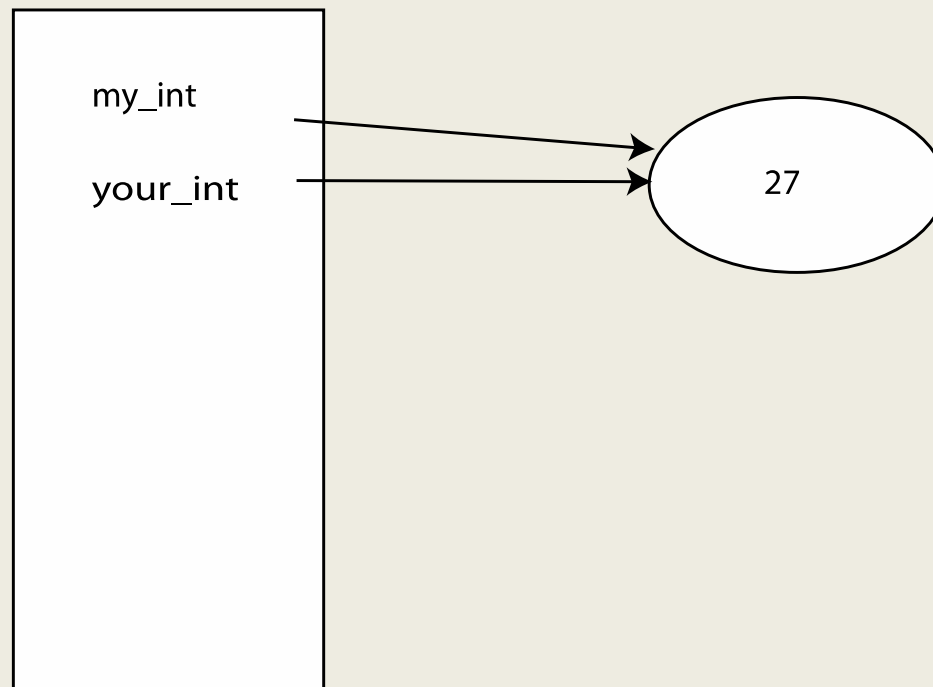
LISTS

```
1 my_int = 27
2 your_int = my_int
```

Two variables are created and they reference the same integer in memory

NameList

Values



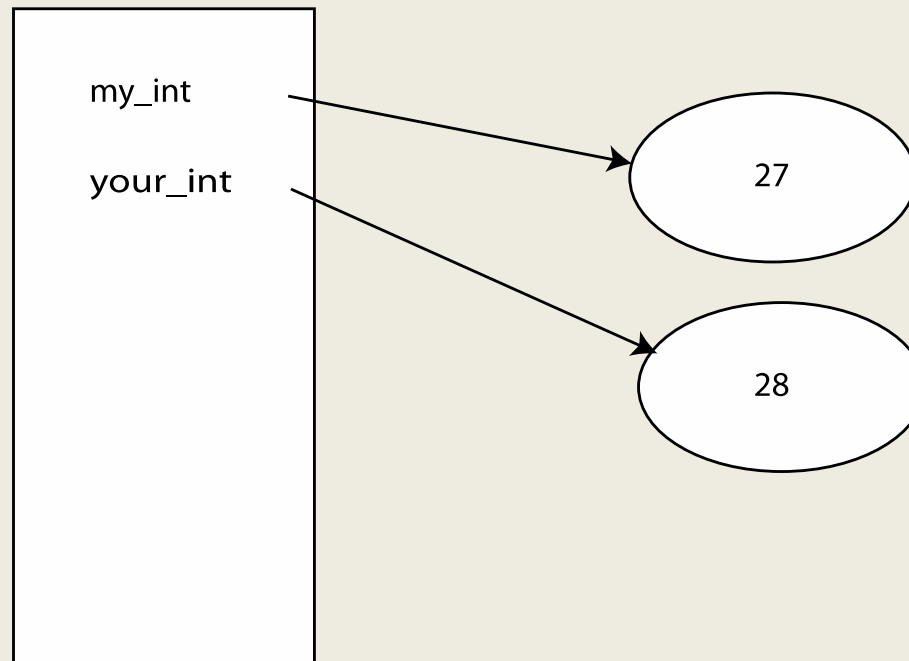
LISTS

```
1 my_int = 27
2 your_int = my_int
3 your_int += 1
```

As soon as there is a change regarding the variable `your_int` a new value is created in memory and the variable `your_int` will reference the new value

NameList

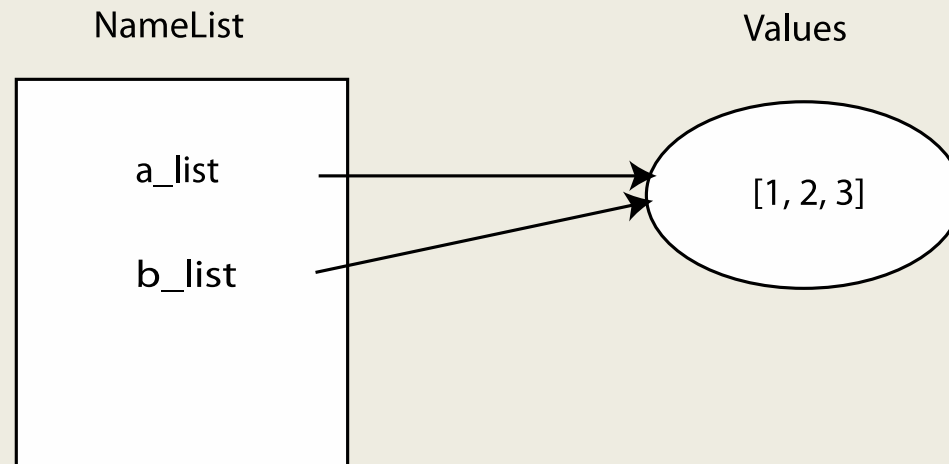
Values



LISTS

- If two variables associate with the same object, then ***both variables will reflect*** any change to that object
- We say that the two variables reference the object

```
1 a_list = [1,2,3]
2 b_list = a_list
```

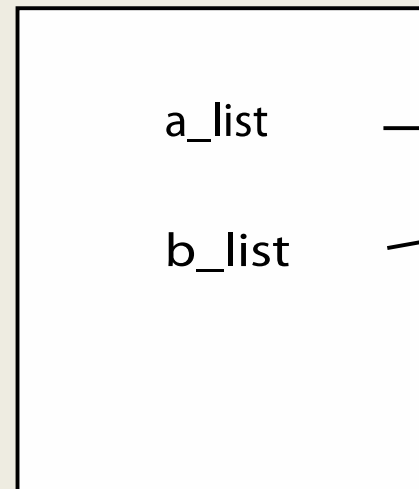


LISTS

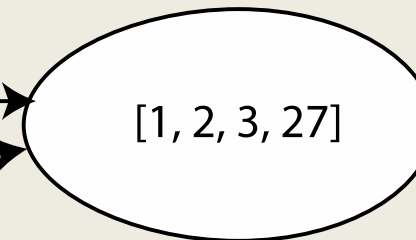
```
1 a_list = [1,2,3]
2 b_list = a_list
3 a_list.append(27)
```

Here we can see that as soon as a change occurs to **a_list** a new object is **not** created! The object simply changes. This is because lists are mutable!

NameList



Values



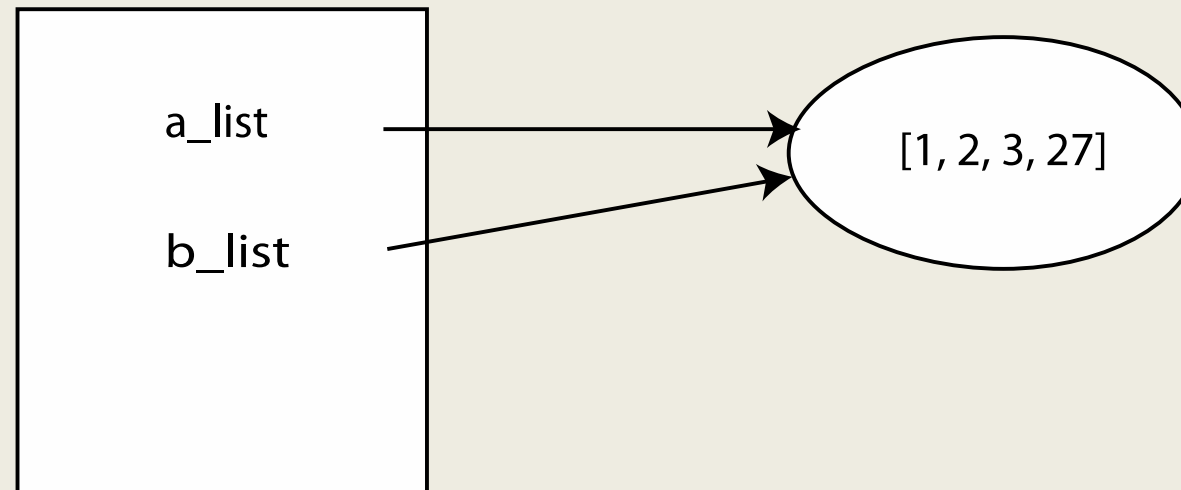
LISTS

When `a_list` and `b_list` are printed the values 1,2,3,27 will be printed two times because the two variables reference the same list!

```
1  a_list = [1,2,3]
2  b_list = a_list
3  a_list.append(27)
4
5  print(a_list)
6  print(b_list)
```

NameList

Values



LISTS

But what if we want to have two separate list instances in memory?

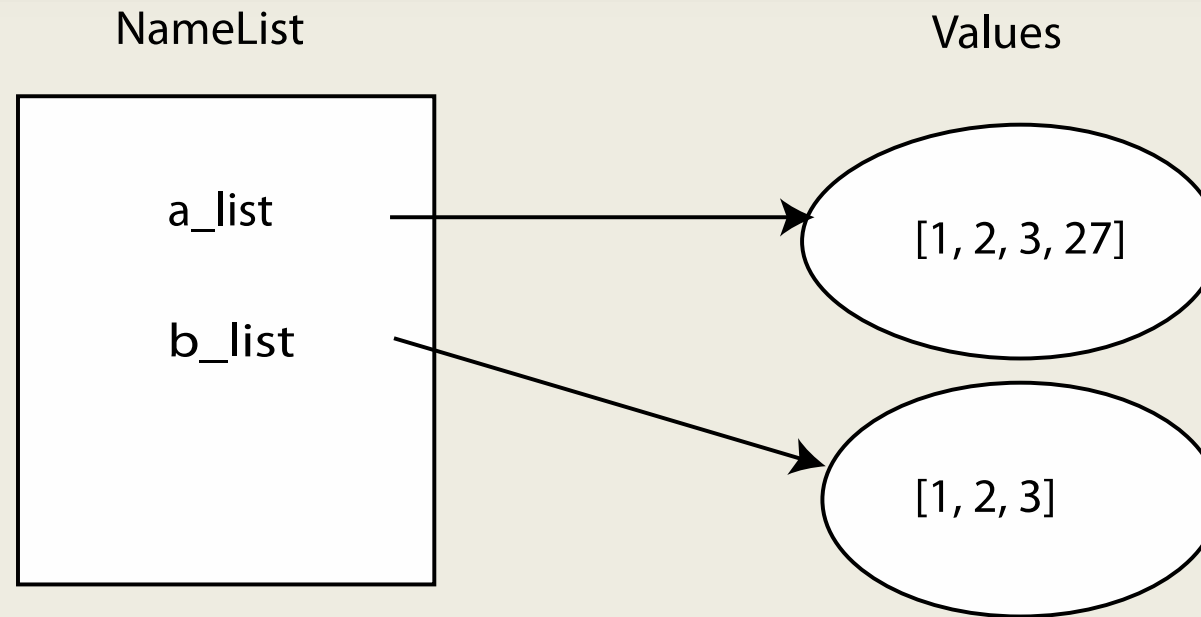
We can copy a list to create a brand new object in memory

```
1 my_list = [1,2,3]
2 your_list = my_list[:]
```

This will create a brand new list in memory. Notice that this is the same syntax as when copying a string!

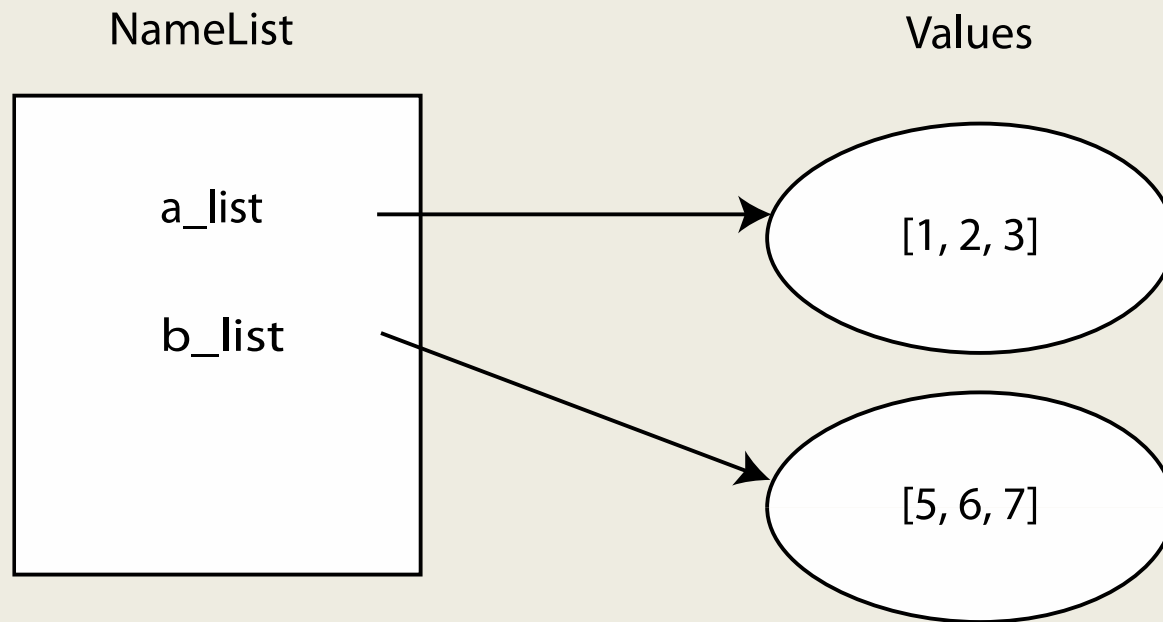
LISTS

```
1 a_list = [1,2,3]
2 b_list = a_list[:] # making a copy of a_list
3 a_list.append(27)
```



LISTS

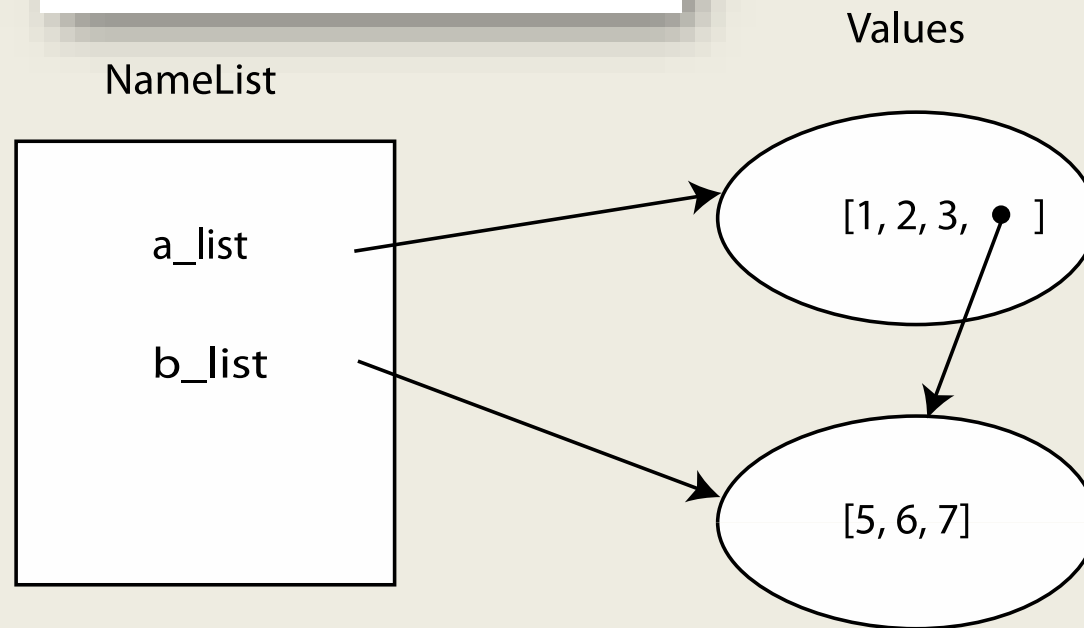
```
1  a_list = [1,2,3]
2  b_list = [5,6,7]
```



LISTS

```
1 a_list = [1,2,3]
2 b_list = [5,6,7]
3
4 a_list.append(b_list)
```

Notice that if we add a `b_list` reference to `a_list` the reference is only added to `a_list` but not the values of `b_list`!

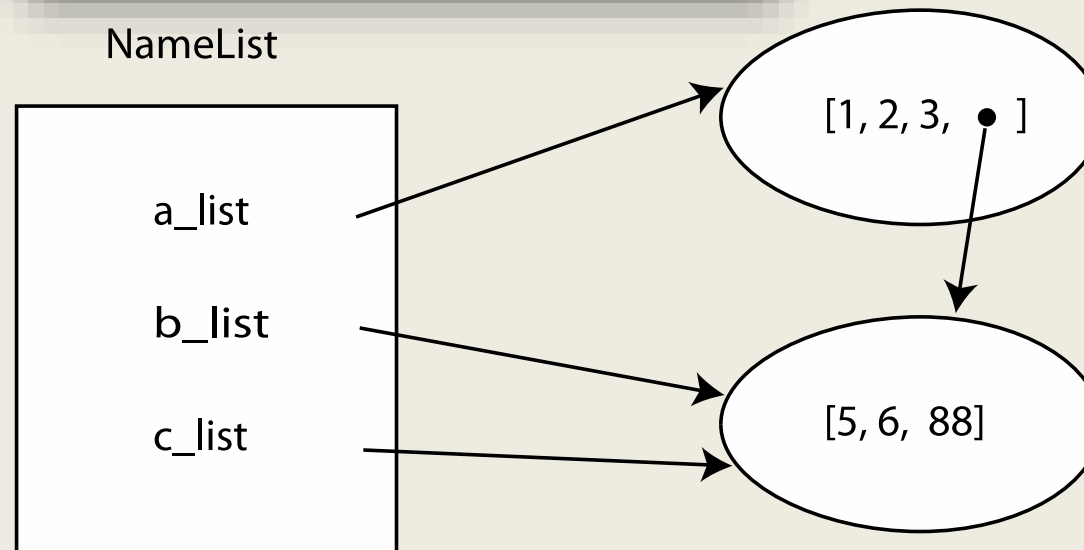


LISTS

```
1 a_list = [1,2,3]
2 b_list = [5,6,7]
3
4 a_list.append(b_list)
5
6 c_list = b_list
7 c_list[2] = 88
```

Values

NameList



LISTS

SHALLOW COPY VS DEEP COPY

- Regular copy
 - the `[:]` approach, only copies the top level reference/association
- if you want a full copy, you can use `deepcopy` from the `copy` module

```
1  import copy
2  a_list = [1,2,3]
3  b_list = [5,6,7]
4
5  a_list.append(b_list)
6
7  c_list = copy.deepcopy(a_list)
8  b_list[0] = 1000
9  c_list[0] = 88
10
11 print(a_list) # [1, 2, 3, [1000, 6, 7]]
12 print(b_list) # [1000, 6, 7]
13 print(c_list) # [88, 2, 3, [5, 6, 7]]
```

LISTS

```
1 import copy
2 a_list = [1,2,3]
3 b_list = [5,6,7]
4
5 a_list.append(b_list)
6
7 c_list = copy.deepcopy(a_list)
8 b_list[0] = 1000
```

