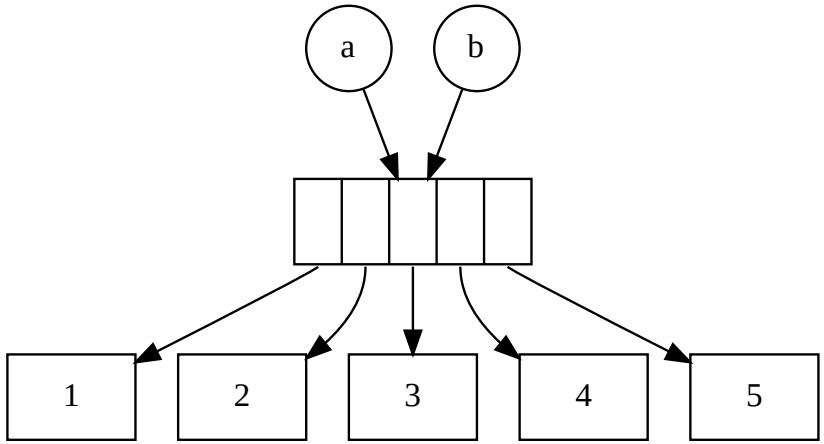
CSC108H Lecture 15

Dan Zingaro

October 15, 2012

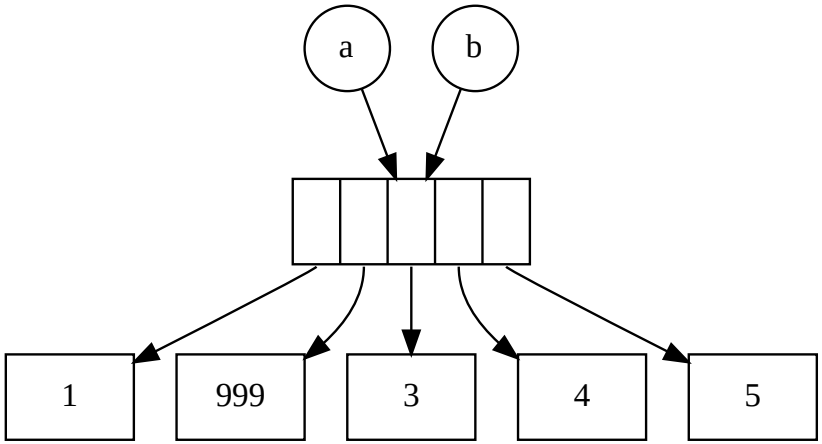
List Mutability

```
a = [1, 2, 3, 4, 5]
b = a # Shared reference!
```



Two variables, but only one list:

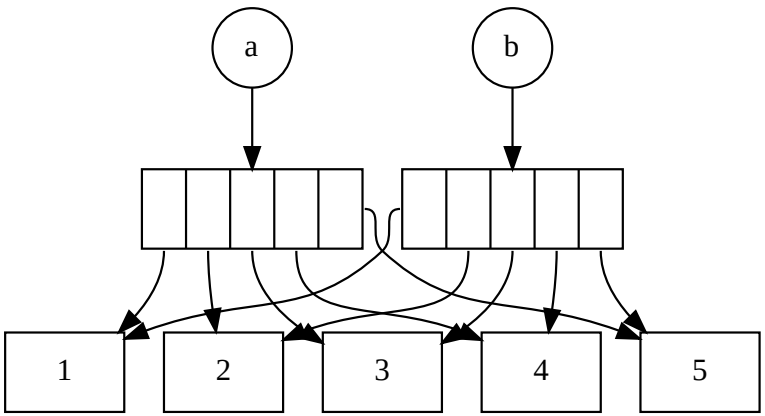
```
b[1] = 999 # check a!
```



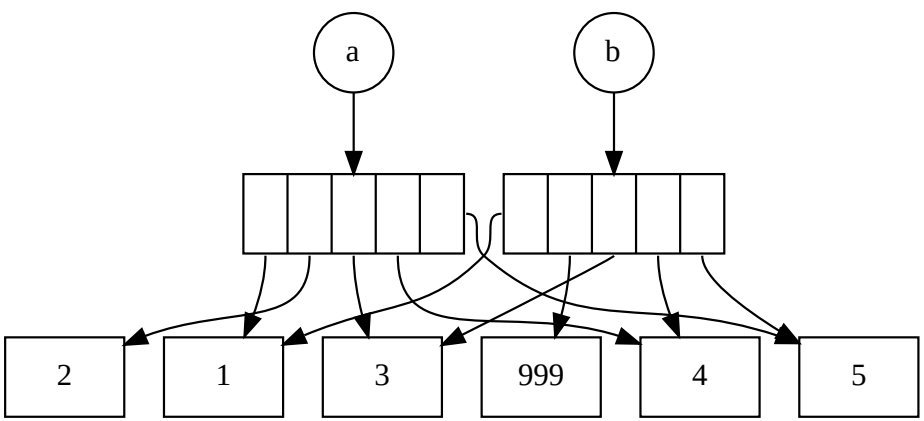
Slice Copies

To avoid the sharing behavior, a second list must be created. Slice syntax creates a new list:

```
a = [1, 2, 3, 4, 5]
b = a[:] # two lists now!
```

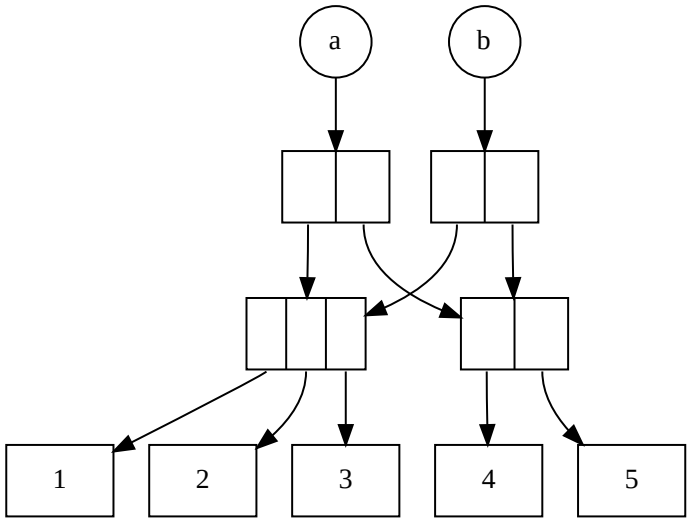



```
b[1] = 999 # a is not modified
```



ConcepTest

```
a = [[1, 2, 3], [4, 5]]
b = a[:]
```



The picture indicates the state after the above code executes. If we then do:

`b.append(8)`

what is the list referred to by a?

- A. `[[1, 2, 3], [4, 5]]` (unchanged)
- B. `[[1, 2, 3], [4, 5], 8]`
- C. `[[1, 2, 3], [4, 5, 8]]`
- D. `[[1, 2, 3], [4, 5], [8]]`

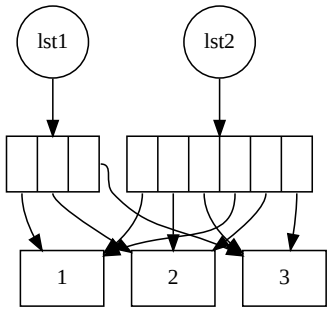
Concatenation and Repetition

- The + operator is overloaded (again) to operate on two lists
- It returns a new list resulting from concatenating the right-hand list to the end of the left-hand list
- It is similar to the `extend` method, but `extend` does not return a new list!
- The * operator takes a list `L` and an integer `n`, and creates a new list by concatenating `n` copies of `L` (like using + `n-1` times)

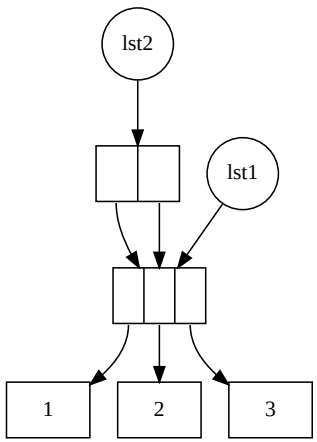
Concatenation and Repetition...

Compare:

```
lst1 = [1, 2, 3]
lst2 = lst1 + lst1
lst2[0] = 99 # lst1 not modified
```




```
lst1 = [1, 2, 3]
lst2 = [lst1] + [lst1]
lst2[0][1] = 99 # lst1 modified!
```



Using range

- The `range` function generates sequences of integers
- If we call `range(n)`, integers from 0 to `n - 1` are generated
- If we call `range(m, n)`, integers from `m` to `n - 1` are generated
- If we call `range(m, n, s)`, integers from `m` to `n - 1`, in increments of `s`, are generated
- Use `list(range(...))` to convert the `range` object to a list for viewing

ConcepTest

What is the list produced by this code?

```
list(range(2, 7, 3))
```

- A. `[2, 5, 8]`
- B. `[2, 5]`
- C. `[2, 5, 7]`
- D. `[2, 3, 4, 5, 6, 7]`

ConcepTest

What is the list produced by this code?

```
list(range(4, 9, 4))
```

- A. `[4, 8]`
- B. `[4, 8, 12]`
- C. `[4, 8, 9]`
- D. `[4, 5, 6, 7, 8, 9]`

Example: Uppercase List and For

Use a for-loop to write both of these:
(1) Given a list of strings, write a function that returns a list that contains all of these strings in uppercase. Do **not** modify the original list.
(2) Given a list of strings, write a function that modifies the list so that all strings are in uppercase. Do **not** return the list. This does not work:

```
def uc_list (lst):
  '''(list of str) -> list of str
  Change lst so that all of its strings are in uppercase.
  '''
  for s in lst:
    s = s.upper()
```

ConcepTest

What is printed by this code?

```
lst = [3, 6, 9]
sum = 0
counter = 0
while counter < len(lst):
  sum += counter
  counter += 2
print(sum)
```

- A. 18
- B. 6
- C. 2
- D. 9
- E. None of the above