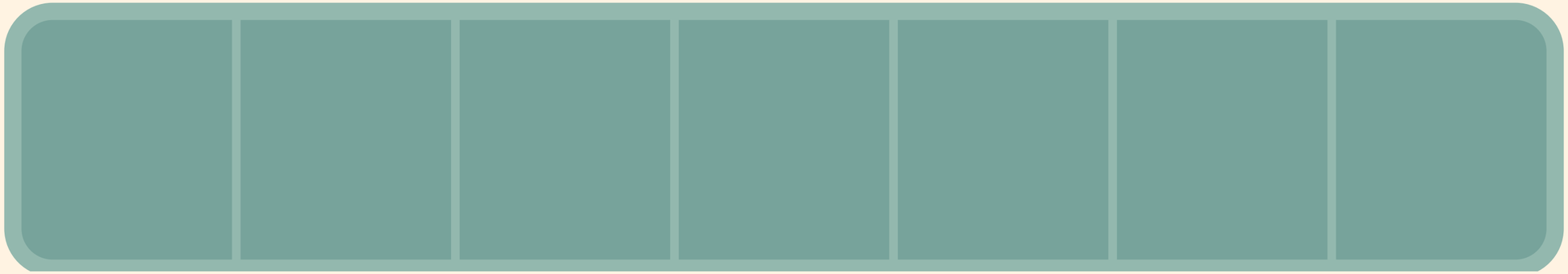
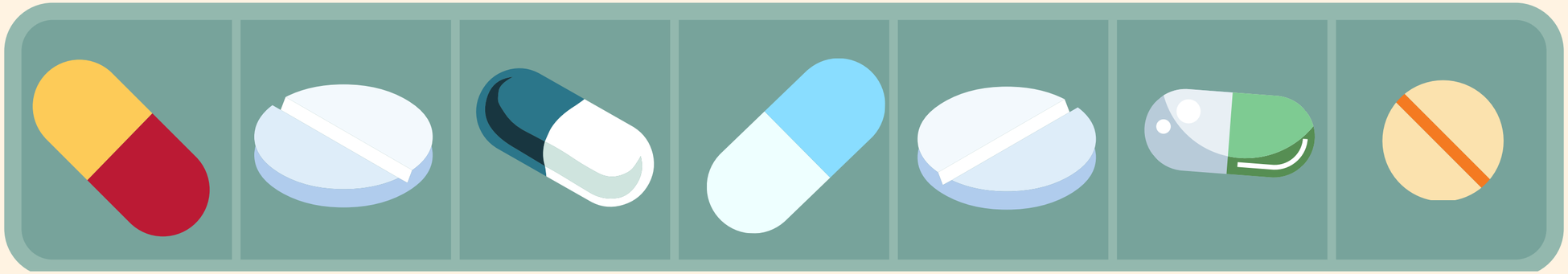
Lists



Lists



Lists

Lists are **ordered collections** of data.
They can hold any of the data types we've seen.

27	'hi'	False	8	0.6	'cat'	-3

Creating Lists

```
tasks = ["Trash", "Dishes", "Laundry", "Dinner"]
```

Creating Lists

```
tasks = ["Trash", "Dishes", "Laundry", "Dinner"]
```

Items

Creating Lists

```
tasks = ["Trash", "Dishes", "Laundry", "Dinner"]
```

Brackets

Creating Lists

```
tasks = ["Trash", "Dishes", "Laundry", "Dinner"]
```

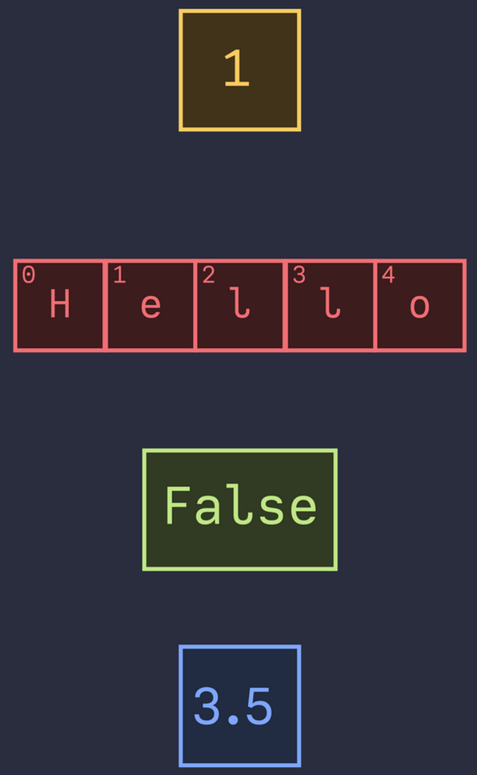
Commas

Script

```
some_list = [1,“Hello”,False,3.5]
```

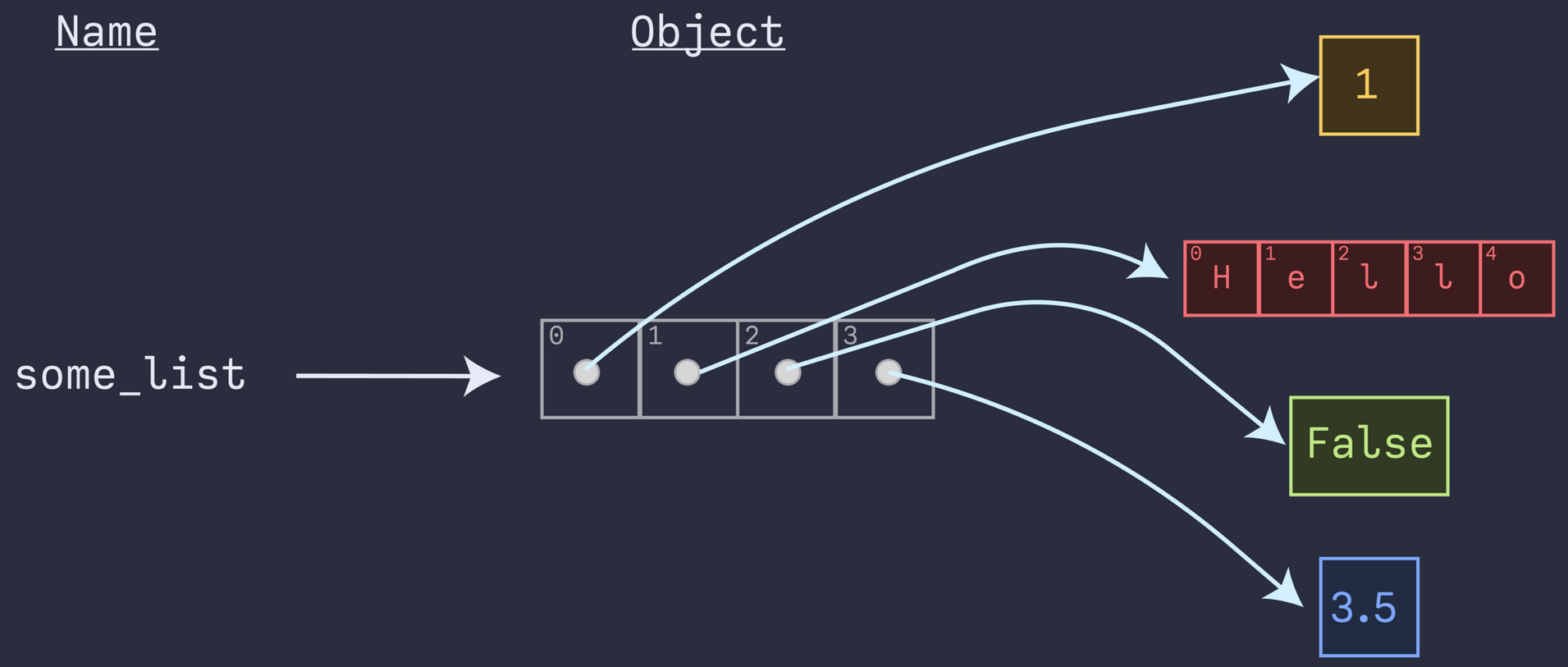
Name

Object



Script

```
some_list = [1,“Hello”,False,3.5]
```

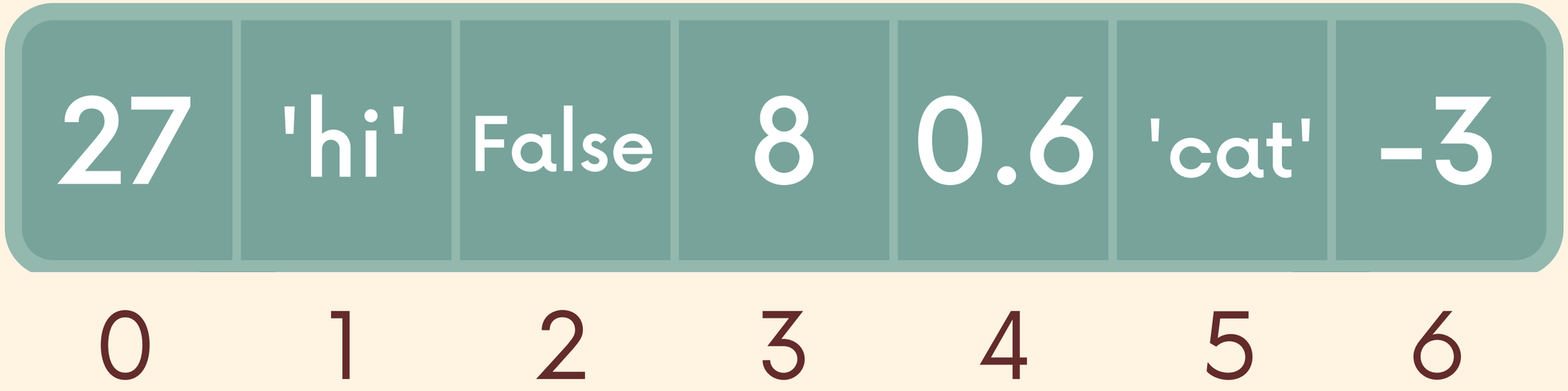


Empty List

```
empty_list = []
empty_list = list()
```

list[index]

We can retrieve individual elements from a list by passing an index number inside of square brackets. Like strings, indices start at 0.


27
'hi'
False
8
0.6
'cat'
-3
0
1
2
3
4
5
6

list[index]

```
> langs = ["Python", "C", "JavaScript"]
> langs[1]
C
> langs[0]
Python
```

Updating

```
> nums = [7,3,9]
> nums[1] = 8
> nums
[7, 8, 9]
```

Update a specific element using its index

[start:stop:step]

```
letters = ['a', 'b', 'c', 'd', 'e']
>>> letters[1:3]
['b', 'c']

>>> letters[0:5:2]
['a', 'c', 'e']
```

Nested Lists

```
> nums = [1, 2, 3, 4, [5,6]]
> nums[4]
 [5,6]
> nums[4][1]
 6
```

Looping Lists

```
> langs = ["Python", "C", "JavaScript","C"]

> for lang in langs:
    print(lang)

Python
C
JavaScript
C
```

index( )

- Returns the index number for the first instance of the value you are passing
- Will give an error if the value is not in the list

```
> langs = ["Python", "C", "JavaScript","C"]

> langs.index("C")

1
```

append

```
> nums = [1, 2, 3, 4]

> nums.append(5)

[1,2,3,4,5]
```

Adds a single value **to the end** of a list

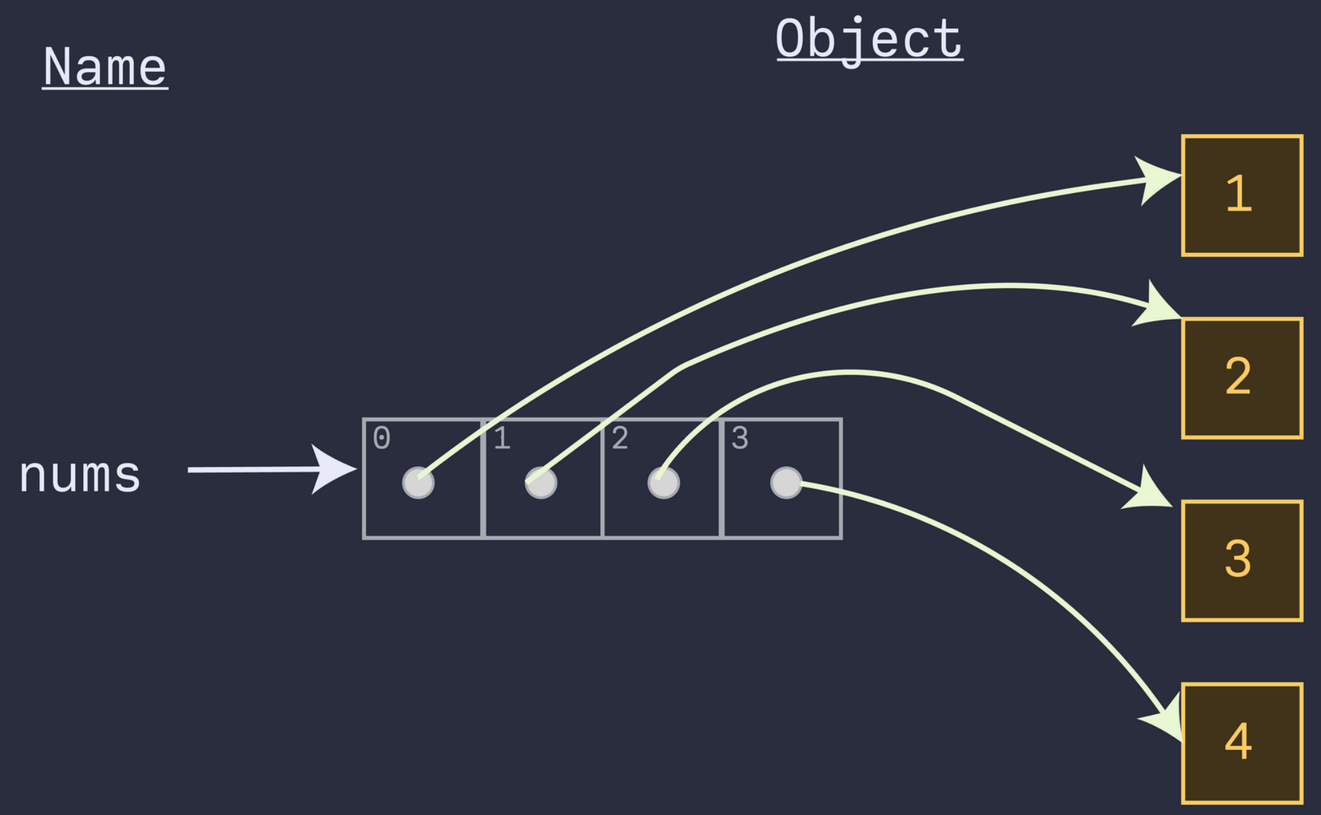
append

```
> nums = [1, 2, 3, 4]
> nums.append(5)
> nums
[1,2,3,4,5]
```

Adds a single value **to the end** of a list

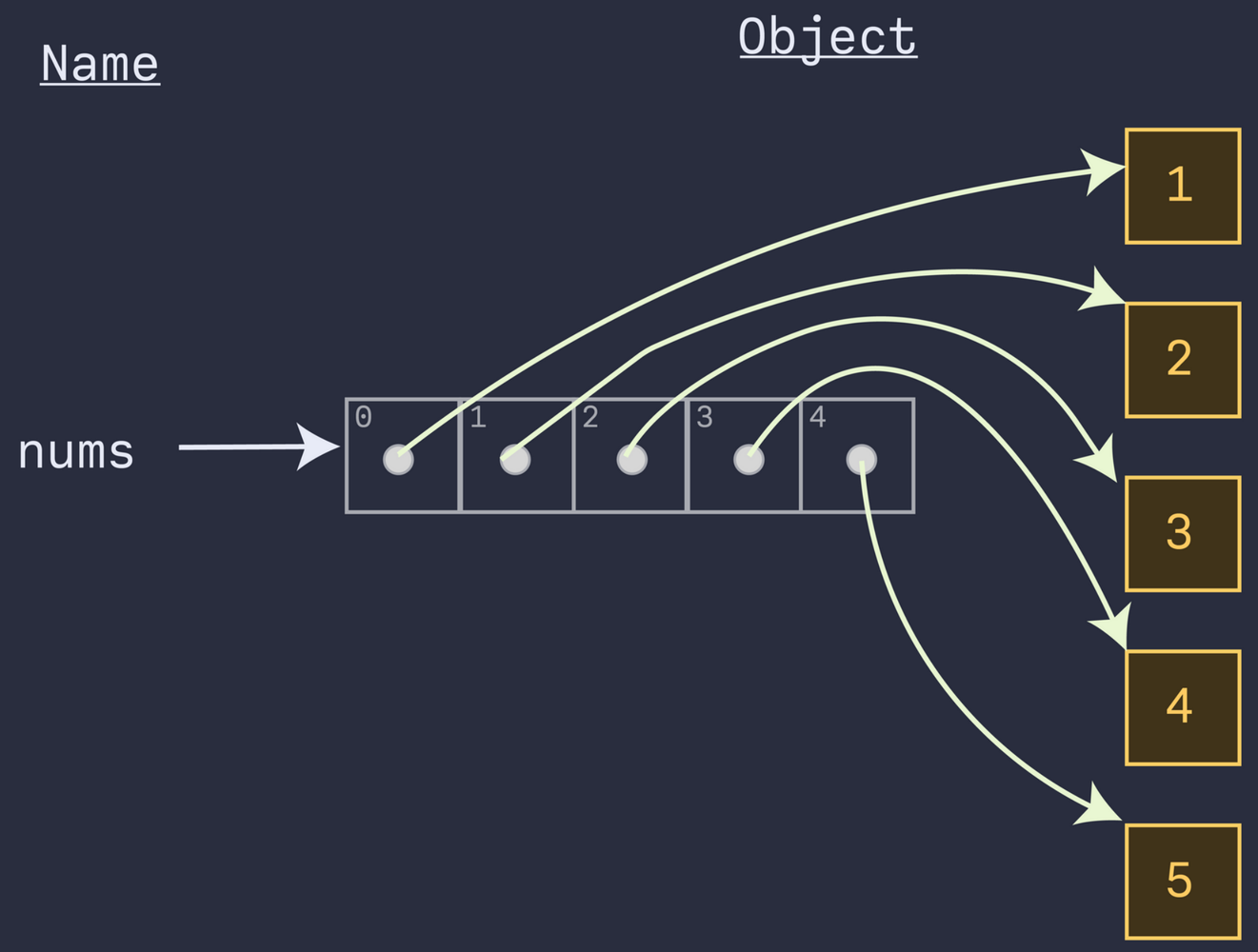
Script

```
nums = [1,2,3,4]
nums.append(5)
```



Script

```
nums = [1,2,3,4]
nums.append(5)
```



extend()

```
> nums = [1,2,3]
> nums.extend("abc")
> nums
[1,2,3,'a','b','c']
```

Accepts an iterable and appends each item from that iterable to the end of the list

element to be
inserted into the list

insert(index, element)

Index before
which to insert
the element

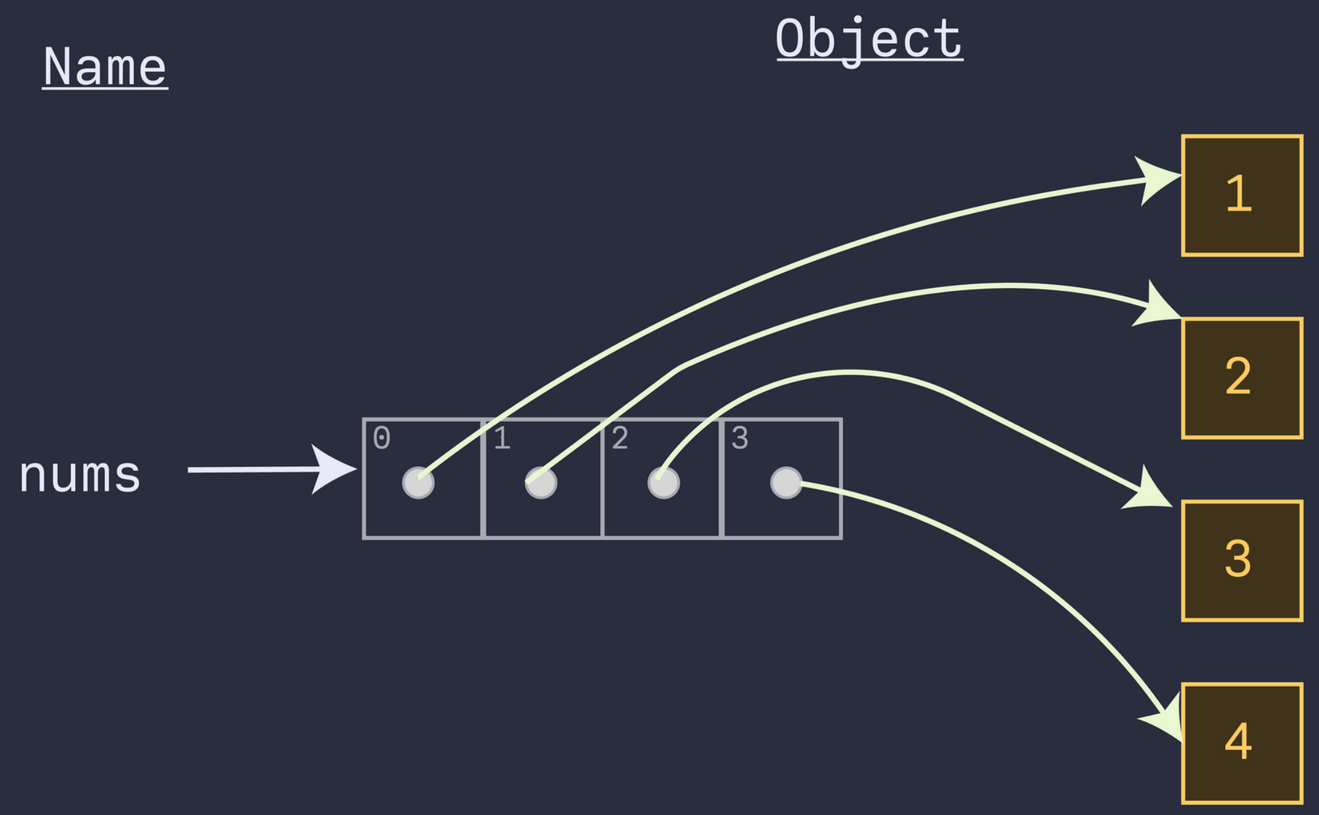
insert()

```
> nums = [7,3,9]
> nums.insert(1,"hi")
> nums
[7, 'hi', 3, 9]
```

Insert "hi" before the element currently located at index 1

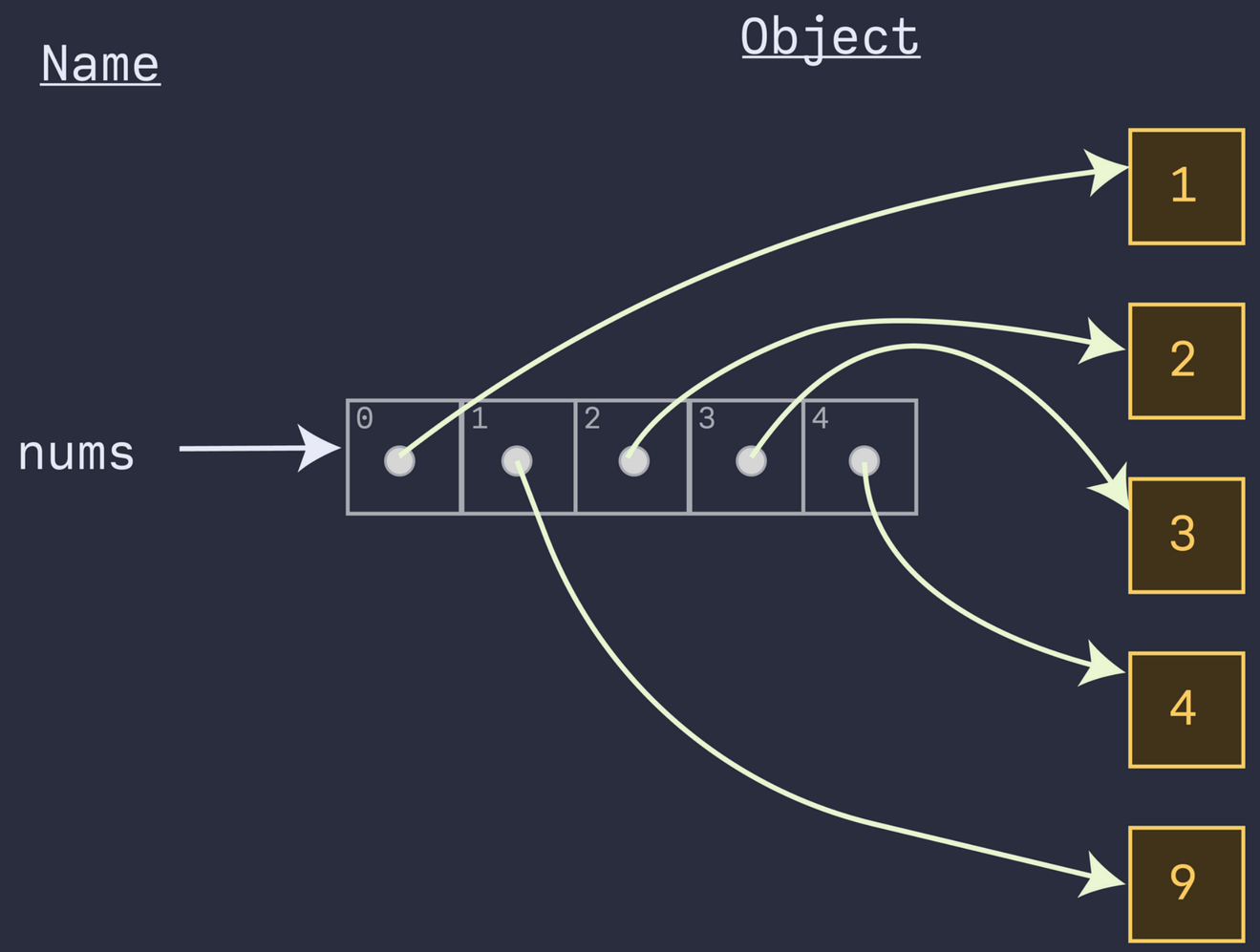
Script

```
nums = [1,2,3,4]
nums.insert(1,9)
```



Script

```
nums = [1,2,3,4]
nums.insert(1,9)
```



list[start:stop:step]

Slices

```
> stuff = ['c',6,'a',9,'t',6]
> stuff[0:2]
['c', 6]
> stuff[0:5:2]
['c', 'a', 't']
```

Addition

```
> [1,2,3] + [4,5,6]
[1, 2, 3, 4, 5, 6]
```

Multiplication

```
> [1,2,3] * 2
[1, 2, 3, 1, 2, 3]
```

Unpacking

```
> user = ["Joe", "Bucky", 42]
> first, last, age = user
> first
 "Joe"
> last
 "Bucky"
> age
 42
```

We can "unpack" values from a list into specific variables.

In this example, the first value in the list is stored in a variable called first.

The second value is stored in a variable called last.

The third value is stored in a variable called age

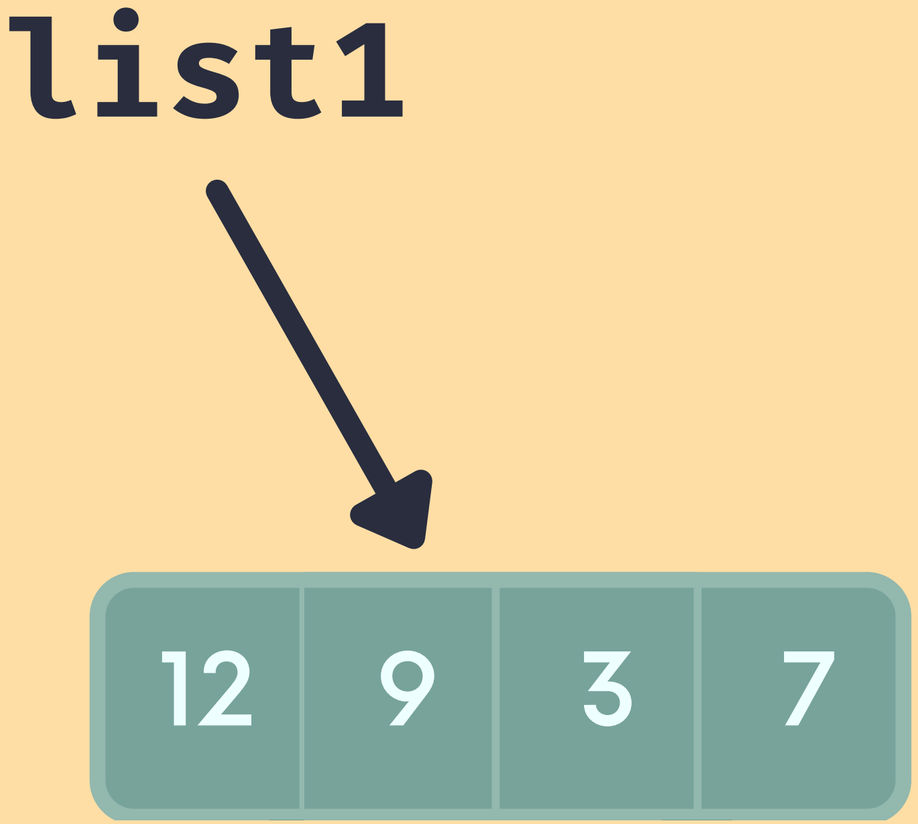
*Unpacking

```
> item = [4, "Pizza", "Plain", 16.98]
> quantity, *others, price = item
> quantity
 4
> others
 ["Pizza", Plain]
> price
 16.98
```

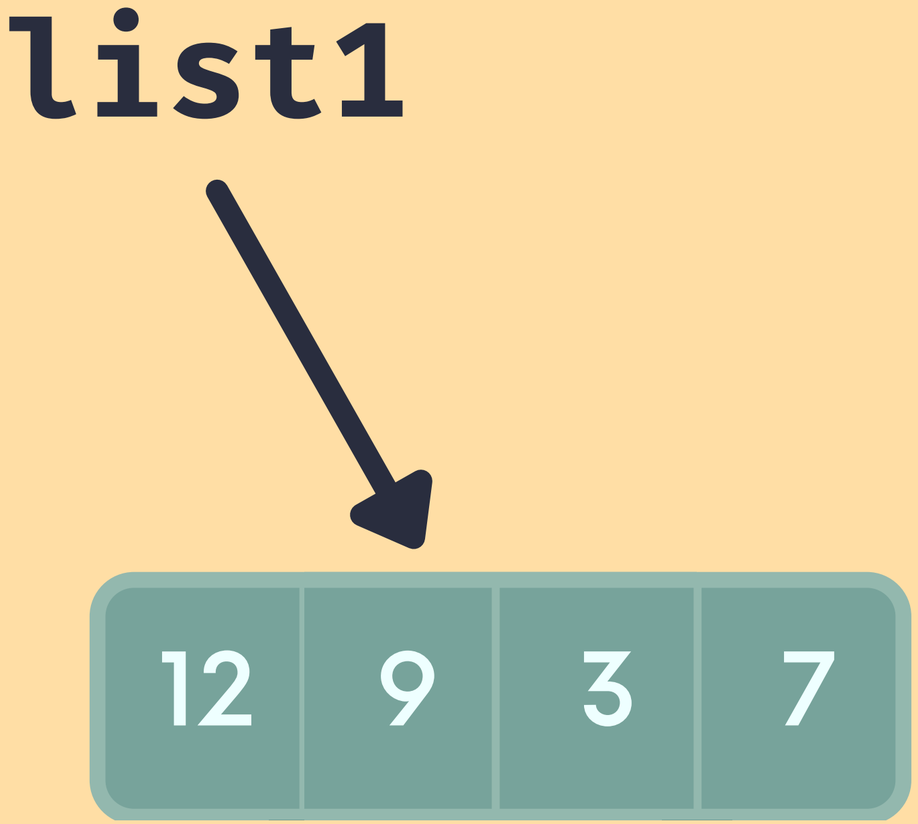
Use an asterisk (*) to gather any remaining unassigned values into a variable.

```
> list1 = [12,9,3,7]
```

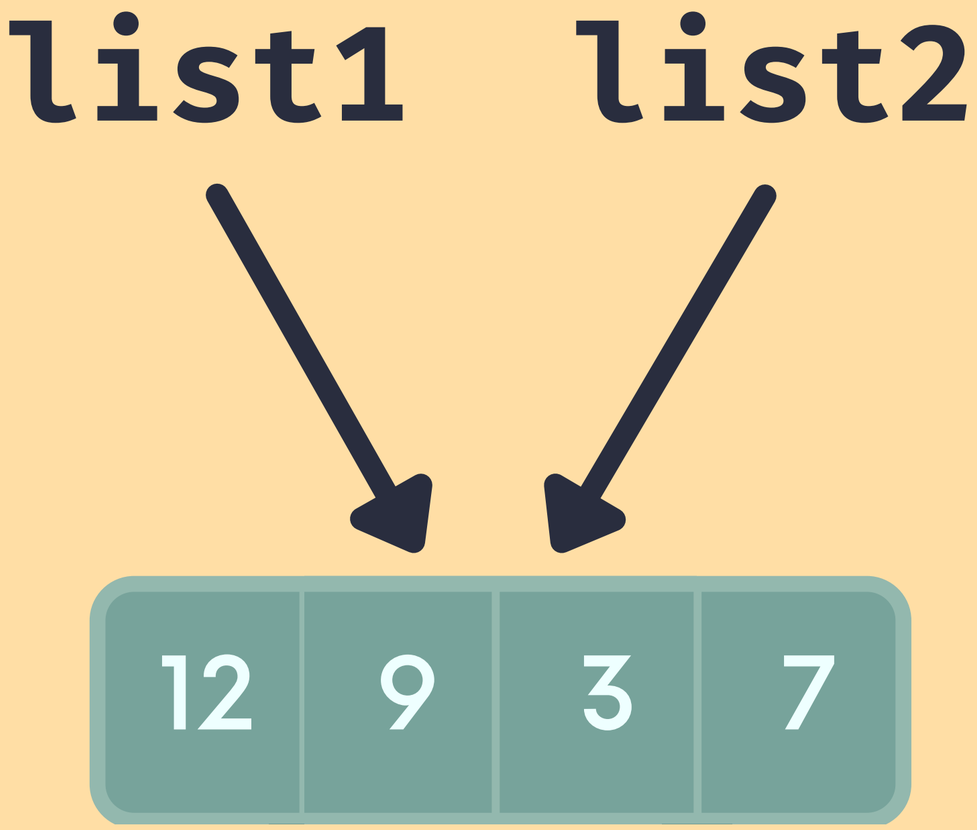
```
> list1 = [12,9,3,7]
```



```
> list1 = [12,9,3,7]
> list2 = list1
```



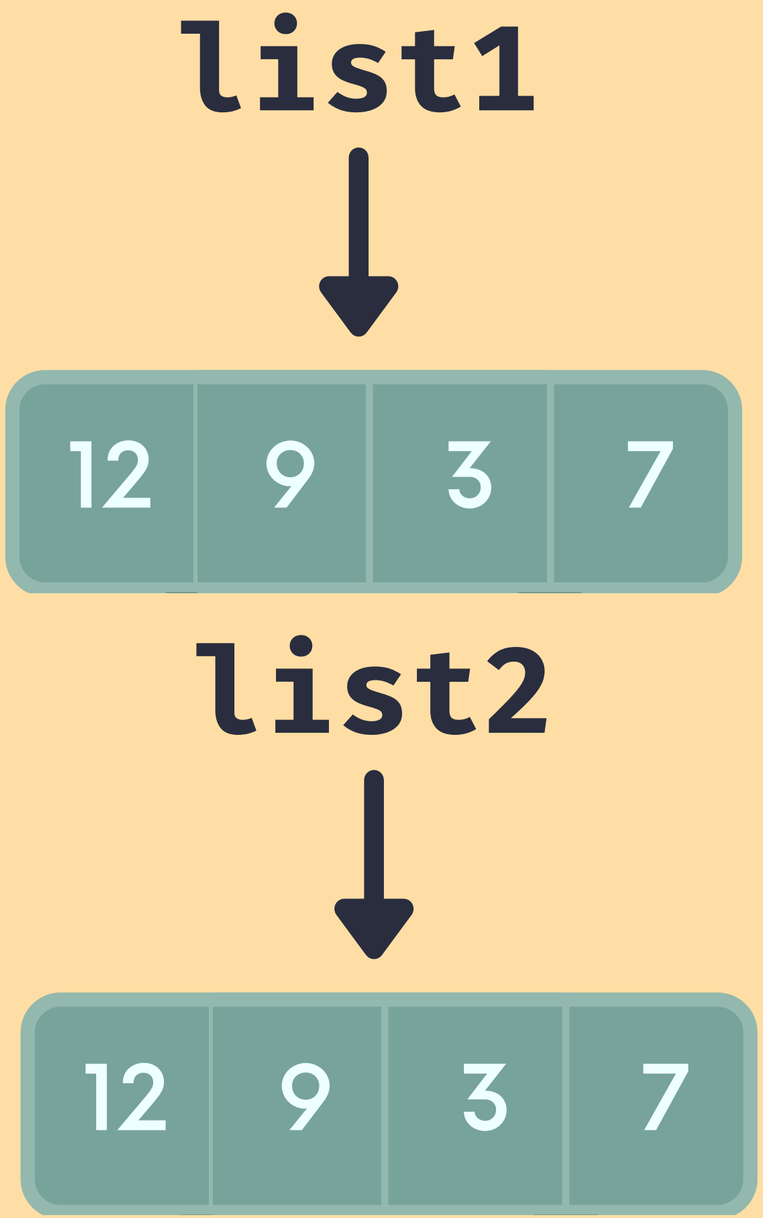
```
> list1 = [12,9,3,7]
> list2 = list1
```



```
> list1 = [12,9,3,7]
```

```
> list1 = [12,9,3,7]
> list2 = [12,9,3,7]
```

```
> list1 = [12,9,3,7]
> list2 = [12,9,3,7]
```



==

```
[1,2,3] == [1,2,3]
True
```

use == to compare the contents inside of two lists. Do they hold the same values?

is

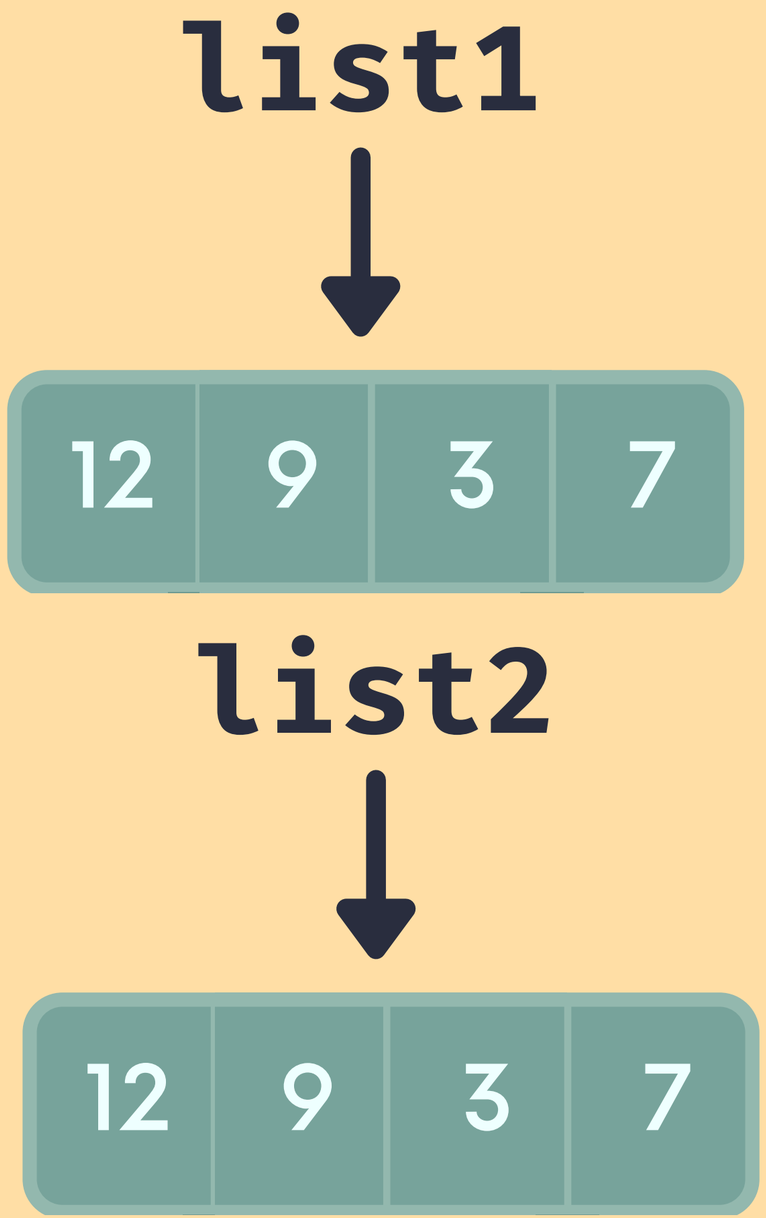
```
[1,2,3] is [1,2,3]
False
```

use **is** to compare the identity of two lists. Are they the same "container" in memory?

copy()

```
> list1 = [12,9,3,7]
> list2 = list1.copy()
```

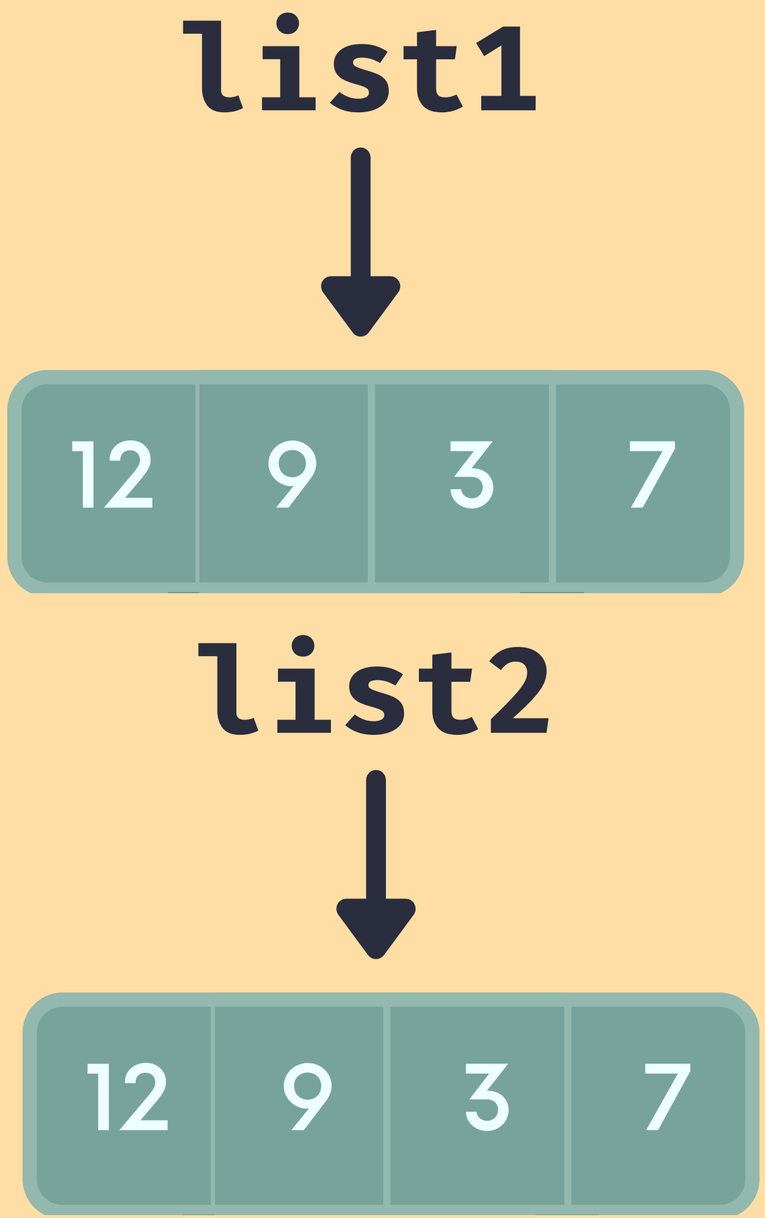
The copy method returns a **shallow copy** of a list. Nested objects are not copied.



Copying with [:]

```
> list1 = [12,9,3,7]
> list2 = list1[:]
```

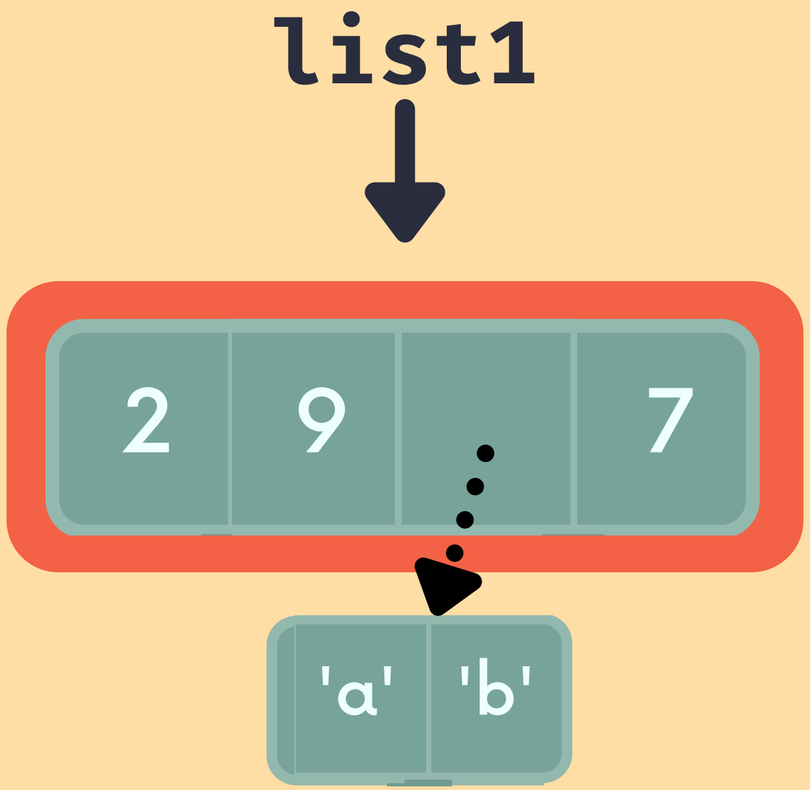
We can also copy lists by creating slices of an entire list. It's not the most readable, but it works!



shallow copy

```
list1 = [2,9,['a','b'],7]
```

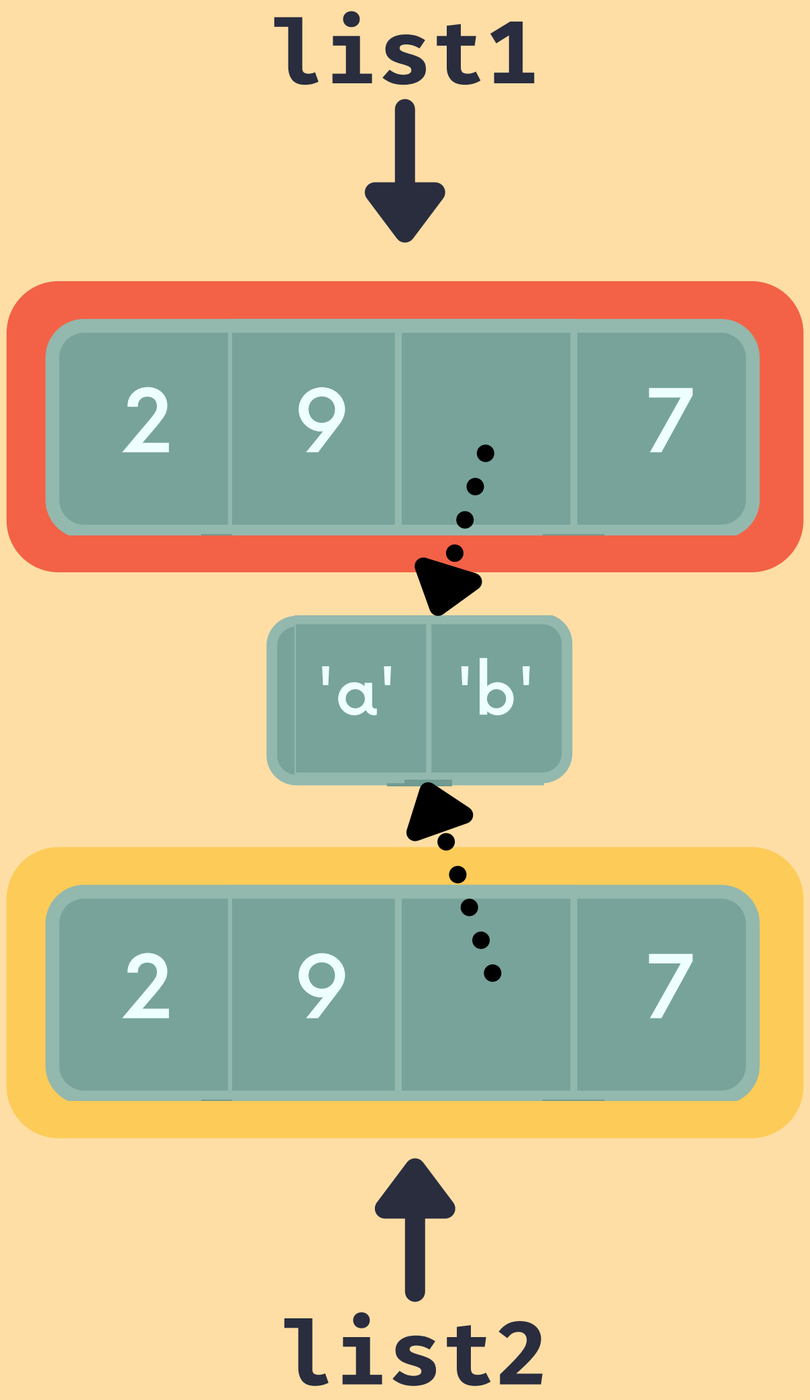
The copy method returns a **shallow copy** of a list. Nested objects are not copied.



shallow copy

```
list1 = [2,9,['a','b'],7]
list2 = list1.copy()
```

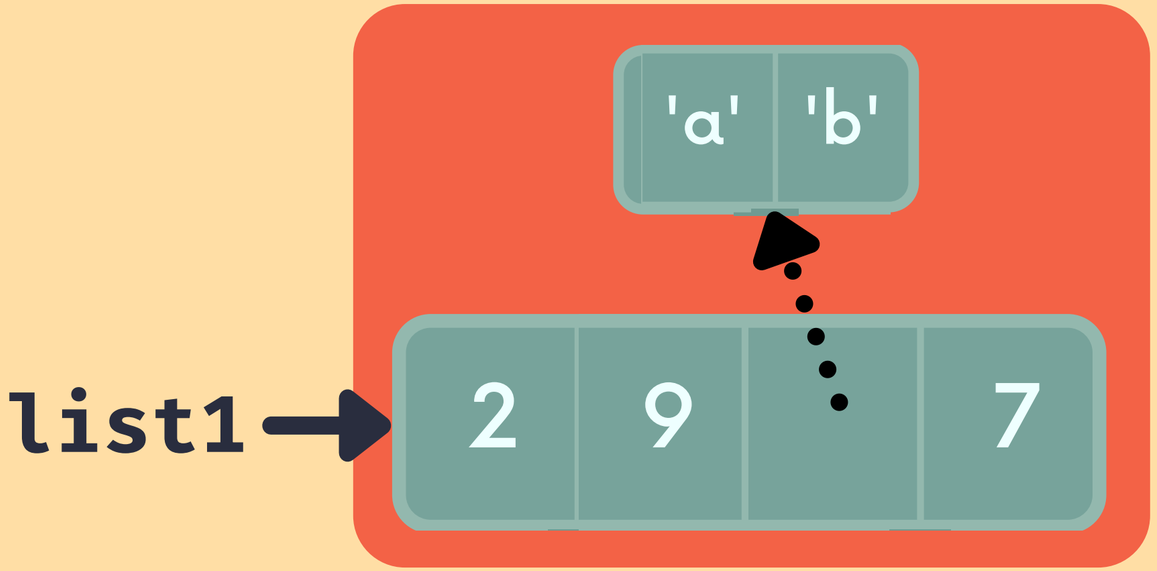
The copy method returns a **shallow copy** of a list. Nested objects are not copied.



deep copy

```
import copy
list1 = [2,9,['a','b'],7]
```

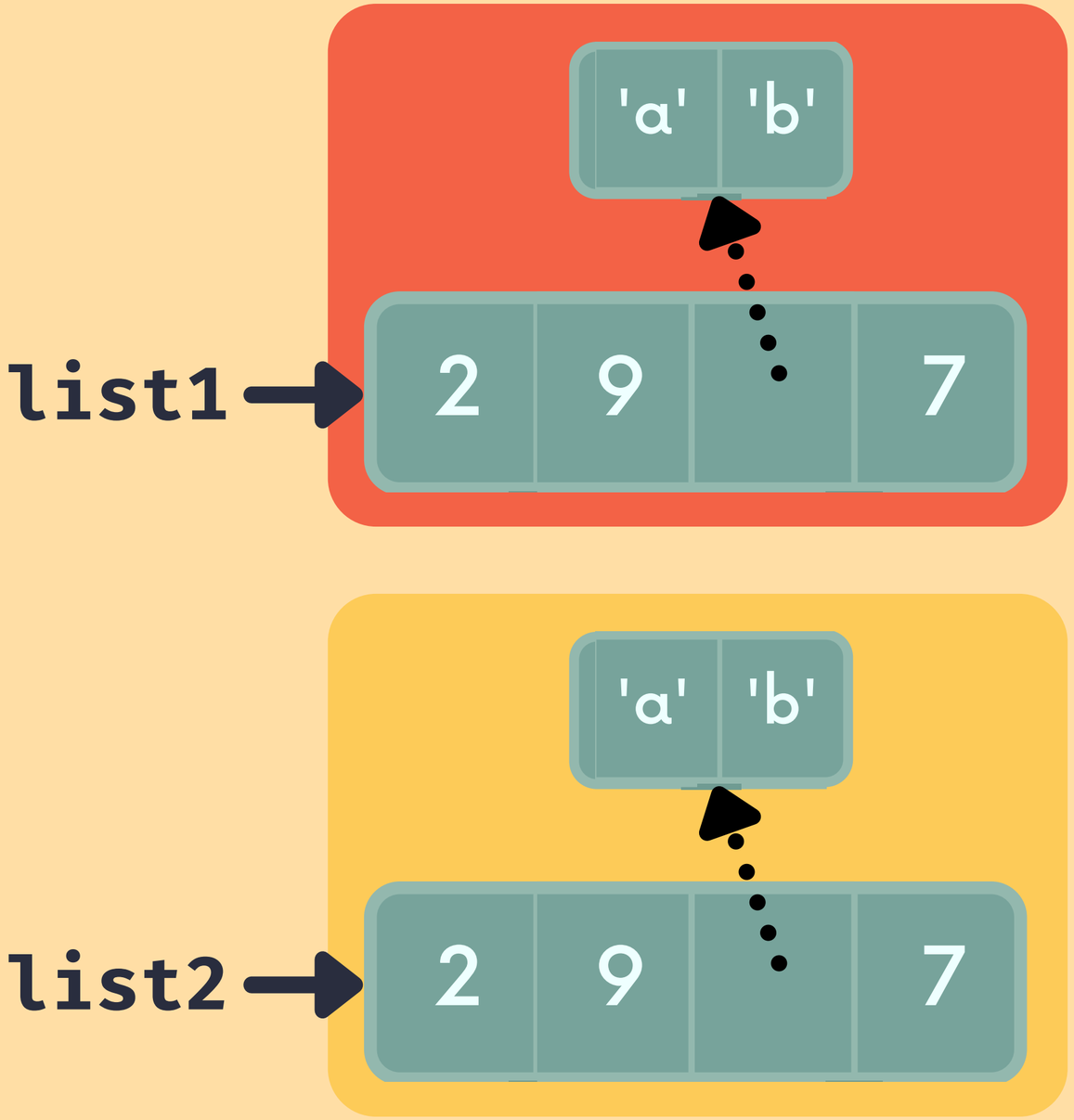
The deepcopy() method will make a copy of a list AND any nested objects contained inside that list.



deep copy

```
import copy
list1 = [2,9,['a','b'],7]
list2 = copy.deepcopy(list1)
```

The deepcopy() method will make a copy of a list AND any nested objects contained inside that list.



clear

```
> langs = ["Python", "C", "JavaScript","C"]
> langs.clear()
> langs
[]
```

the clear() method removes all items from a list

remove

```
> langs = ["Python", "C", "JavaScript","C"]
> langs.remove("C")
> langs
[Python, JavaScript, C]
```

The remove(x) method will remove the FIRST element in the list that has a value of x

pop

```
> langs = ["Python", "C", "JavaScript","C"]
> langs.pop()
'C'
> langs
['Python', 'C', 'JavaScript']
```

The pop() method removes AND returns the last element from a list.

pop(idx)

```
> langs = ["Python", "C", "JavaScript","C"]
> langs.pop(0)
'Python'
> langs
['C', 'JavaScript', 'C']
```

pop() also accepts a specific index. It will remove the item at that index in the list AND return it

del

```
> langs = ["Python", "C", "JavaScript","C"]
> del lang[2]
> langs
[Python, C, C]
```

The del statement (it's not a method!) can be used to delete an item from a specific index in a list

count

```
> langs = ["Python", "C", "JavaScript","C"]

> lang.count("C")

2
```

The count method returns the number of times a value occurs in a list. If the value is not in the list, it returns 0

reverse

```
> nums = [1,2,3,4,5]
> nums.reverse()
> nums
[5, 4, 3, 2, 1]
```

the reverse() methods reverses a list **in-place**

sort

```
> nums = [2,8,1,9,3]
> nums.sort()
> nums
[1, 2, 3, 8, 9]
```

the reverse() methods reverses a list in-place

join

```
> fruits = ["Apple", "Kiwi", "Pear"]
> " ".join(fruits)
'Apple Kiwi Pear'

> "!!!".join(fruits)
'Apple!!!Kiwi!!!Pear'
```

join() is a string method that joins together the elements of an iterable into a single string. Whatever string you call it on will be used as a separator.

split

```
> birthday = "03/27/2020"
> birthday.split("/")
['03', '27', '2020']
```

split() is a string method that will split a string on a given character. It returns a list that holds the split strings.