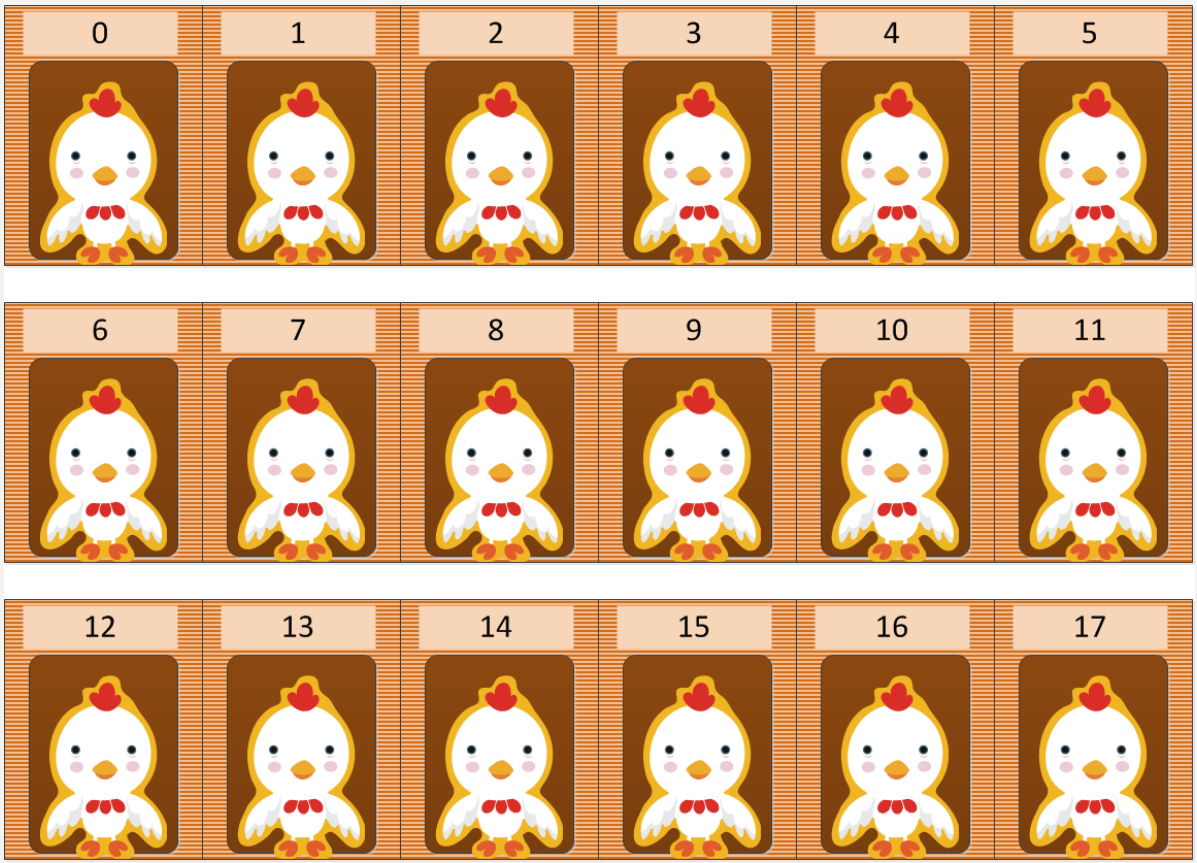
ENGG1330 Computer Programming I

6. ARRAY

VICTOR LEE

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING



chicken[0]…
chicken[1]…
chicken[2]..

ARRAY

- **Ordered** list of data (also called items or elements)

```
weight=[2.1, 1.5, 1.1, 0.8, 0.7]
```

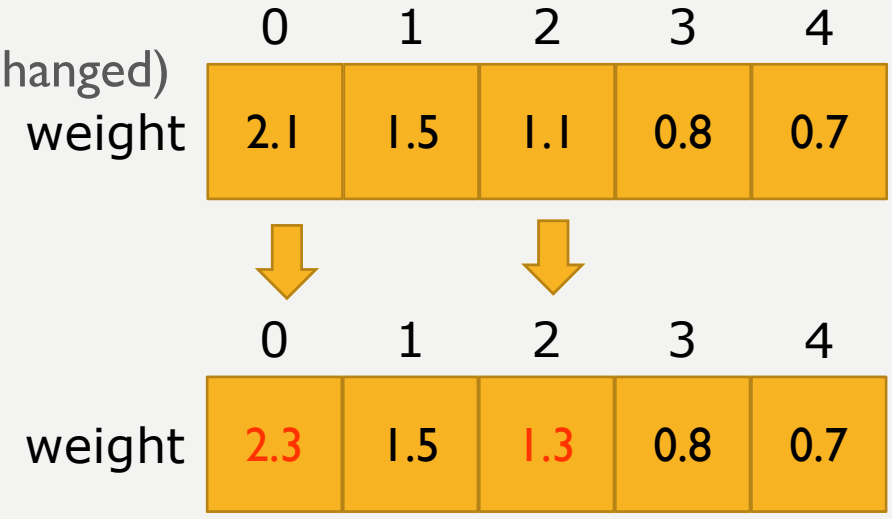
- Can be accessed by specifying the **index** inside the brackets
  - Index is an **offset** from the beginning of the list
  - The offset of the first item is **zero**

```
print(weight[0])
print(weight[2])
```

The indices start at 0

- It is **mutable** (items can be changed)

```
weight[0]=2.3
weight[2]=1.3
```



ITERATE ITEMS IN AN ARRAY

- Items in an array can be iterated using a `for` loop and the built-in function `range()`

```
weight = [2.1, 1.5, 1.1, 0.8, 0.7]
for i in range(5):
  print(weight[i])
```

`range` returns a list of indices from 0 to 4

- Size of an array can be obtained by the built-in function `len()`

```
weight = [2.1, 1.5, 1.1, 0.8, 0.7]
for i in range(len(weight)):
  print(weight[i])
```

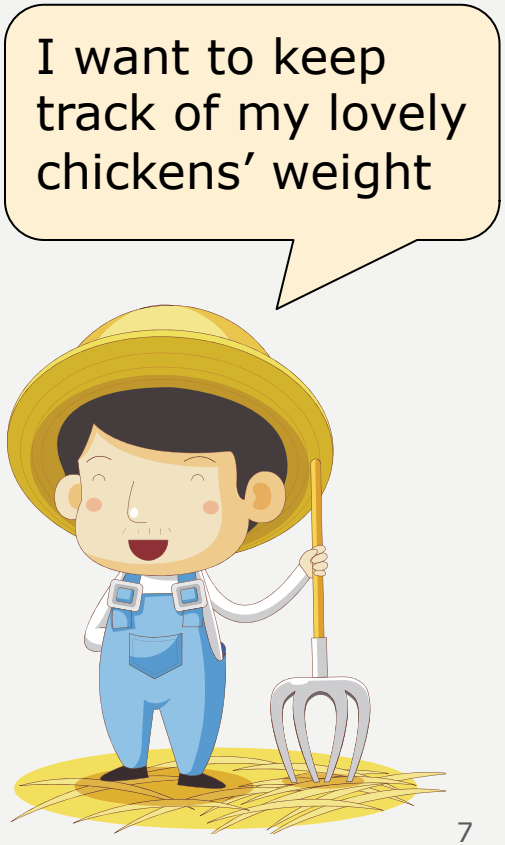
ITERATE ITEMS IN PYTHON

- In Python, we can use `for in` loop to iterate over the sequence directly.
- The items in the list will be assigned to the iterating variable `item` one-by-one.

```
weight = [2.1, 1.5, 1.1, 0.8, 0.7]
for item in weight:
  print(item)
```

PETER'S WISH LIST TO A COMPUTER ENGINEER

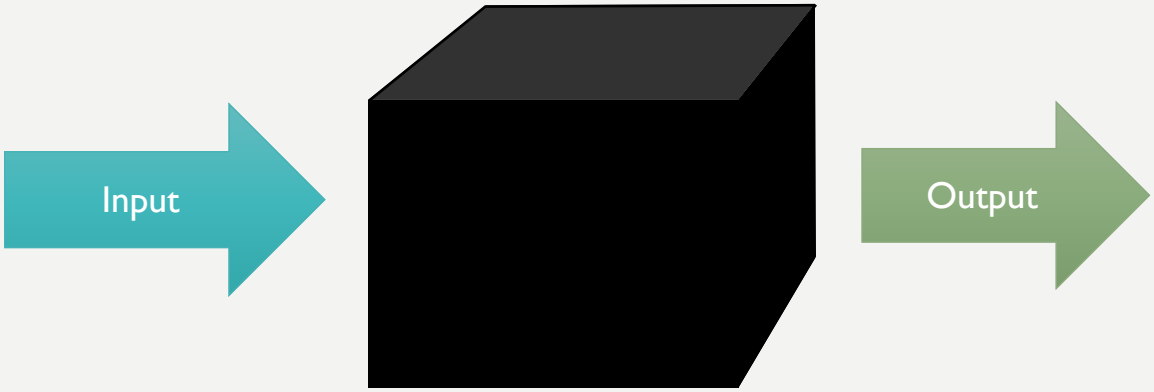
- Input the weights of chicken every day
- Locate the lightest chicken
- Locate the heaviest chicken



WHAT IS THE WISH LIST OF THE COMPUTER ENGINEER?

- How many chicken Peter has (will have?), fixed or unknown?
- In what way the weights will be entered?
- What is the expected output of "locating the lightest chicken"?
- What is the expected output of "locating the heaviest chicken"?

BLACK BOX



MAIN PROGRAM

```
weights=getInput()

lightest=findLightest(weights)

heaviest=findHeaviest(weights)

print(" The lightest chicken is at "+ str(lightest))

print(" The heaviest chicken is at "+ str(heaviest)+"!  Cut its food!")
```

THE TRUTH IS:

- In Python, there is no built-in support for Arrays
- The closest implementation is **List**
- Python List:
  - Ordered collection of object type
  - List can contain any sort of objects: number, string, list

LIST CREATION

```
#create an empty list
l=[]

#create a list with 3 integers
l=[1, 2, 3]

#create a list with items of different types, even a list
l=['ENGG', 1330, ['Computer', 'Programming']]

#create a list with repeating items using the '*' operator
l=[0]*10

#create a list by concatenating existing lists using the '+'
operator
l=[1, 2, 3] + [4, 5, 6]

#create a list using built-in function list()
l=list(range(2,20,2))
```

enclose items in square brackets, separated by commas

GET INPUT

```
def getInput():
    weights=[]
    print("How many chickens do you have?")

    n=int(input())
    print("Please enter their weights one by one")
    for i in range(n):
        x=float(input())
        weights.append(x)

    return weights
```

`append` is a list method used to add a new item to the end of a list

Function vs. method

- Similar: both take arguments and return a value
- Different: syntax
  - function syntax: `func(arg)`
  - method syntax: `weights.append(x)`
    - A method call is called invocation, i.e., invokes `append()` on `weights`

```
weights=getInput()
print(weights)
```

FIND THE LIGHTEST CHICKEN

```
def findLightest(weights):
    min=weights[0]
    index=0
    for i in range(len(weights)):
        if min>weights[i]:
            min=weights[i]
            index=i
    return index
```

```
l=findLightest(weights)
print("The lightest chicken is located at "+str(l))
```

FIND THE HEAVIEST CHICKEN

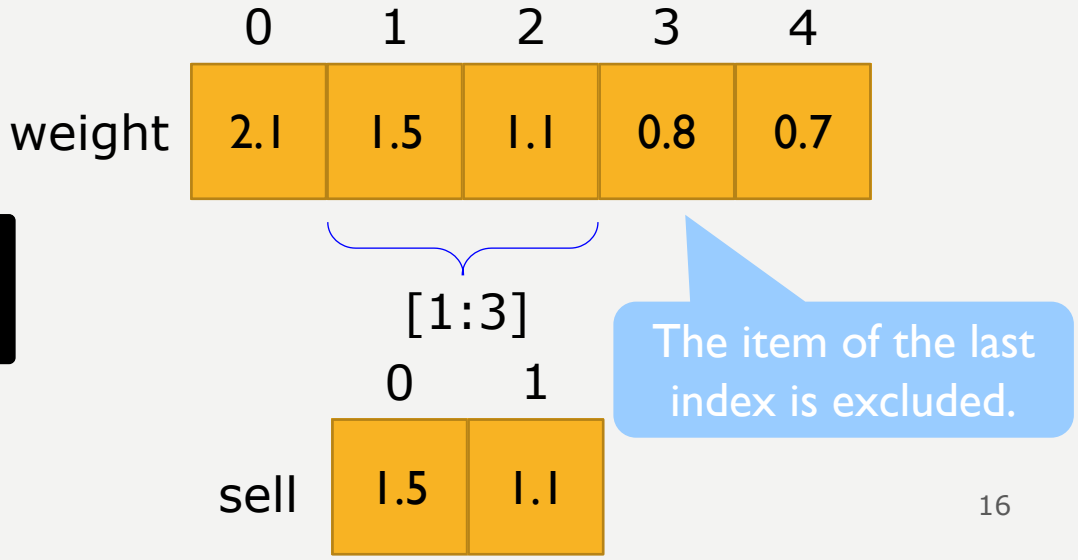
```
def findHeaviest(weights):
    max=weights[0]
    index=0
    for i in range(len(weights)):
        if max<weights[i]:
            max=weights[i]
            index=i
    return index
```

```
h=findHeaviest(weights)
print("The heaviest chicken is located at "+str(h))
```

SLICE A LIST

- A list can be sliced from another list by specifying the first index, the last index (exclusive) and the step (increment between indices)

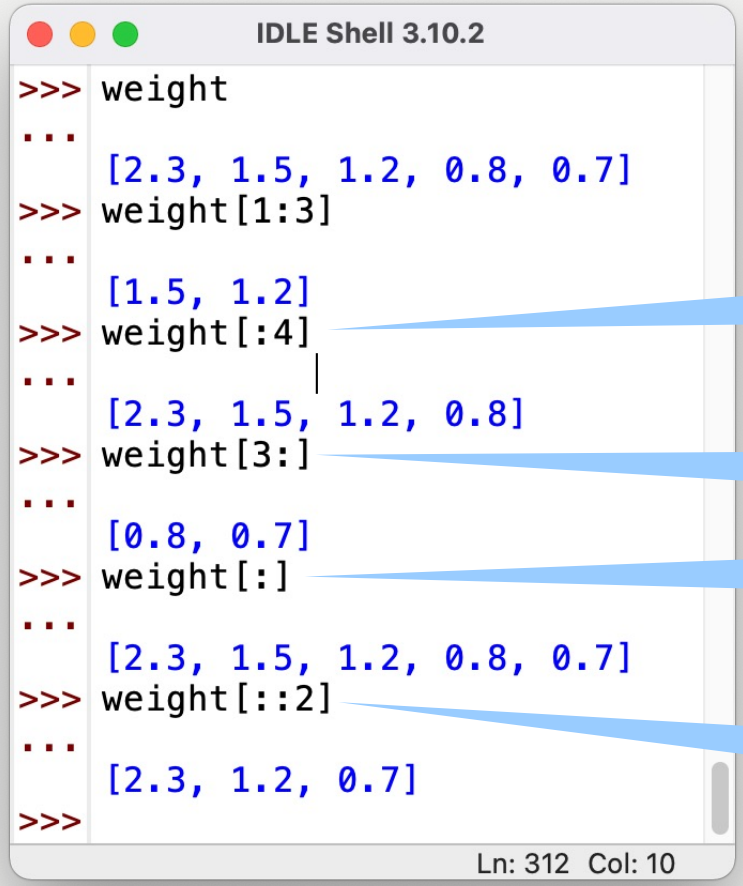
```
List[first_index:last_index:step]
```




```
sell = weight[1:3]
```

SLICE A LIST

	0	1	2	3	4
weight	2.3	1.5	1.2	0.8	0.7



Omit the first index, the slice starts at the beginning

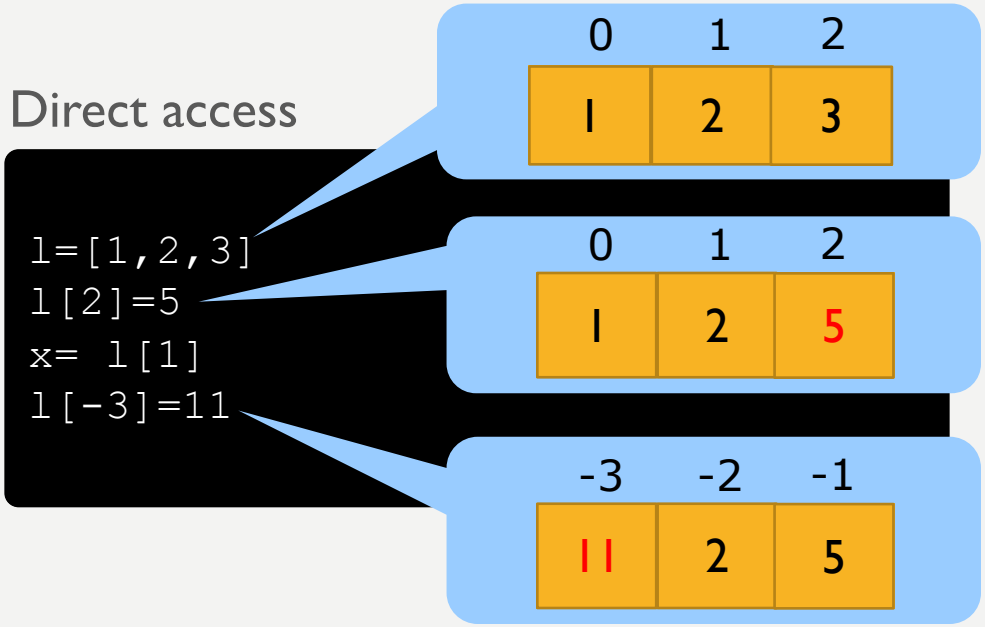
Omit the second index, the slice goes to the end

Omit both, the slice is a copy of the original list

"step" specifies the increment between indices

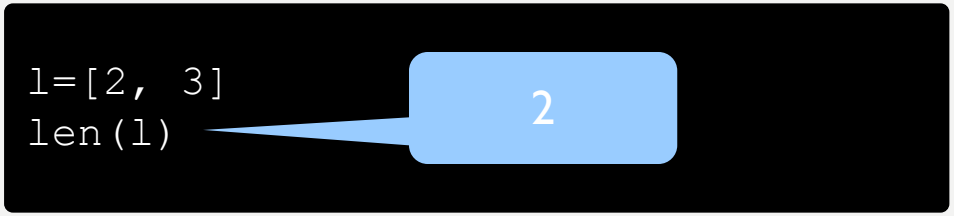
ACCESS THE CONTENT

- Direct access



Index can be negative and the index of -1 refers to the last item in the list.

- Get the size of a list



ACCESS THE CONTENT

- Test if a particular item is in the list using the `in` operator

```
data = [1, 3, 2, 8, 6, 11, 3, 4]

print( 3 in data)

print( 7 in data)
```

True

False

ACCESS THE CONTENT

- Find the (first) index of a given item in a list (if present), using the `index` method

```
list.index(item_to_search [,start,[end]])
```

```
data = [1, 3, 2, 8, 6, 11, 3, 4]

data.index(3)
data.index(8)
data.index(3,2)
data.index(3,2,7)
```

1

3

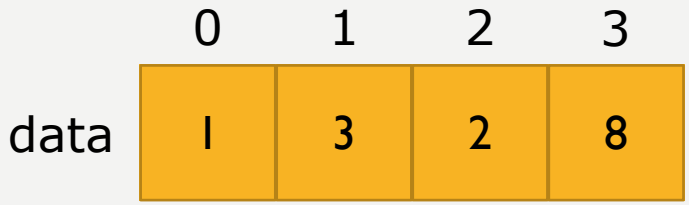
6

6

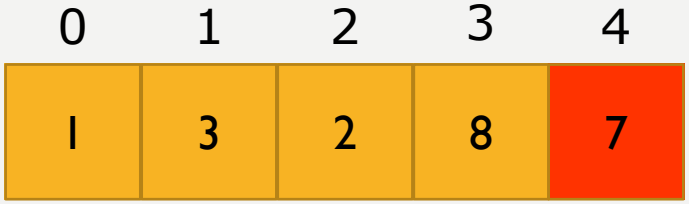
DYNAMIC NATURE OF LIST

- Insert item into a list

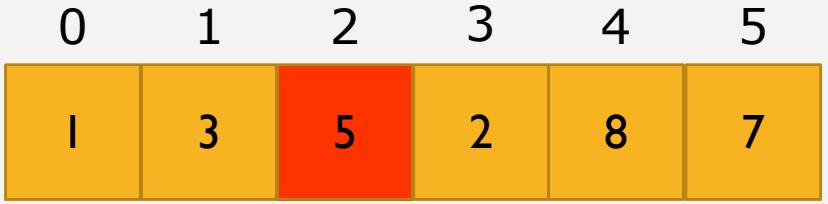
```
#insert an item at the end
list.append(item)


#insert an item into index position
list.insert(index, item)
```



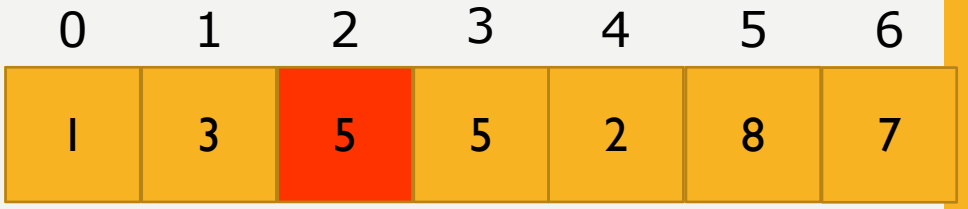

```
data.append(7)
```



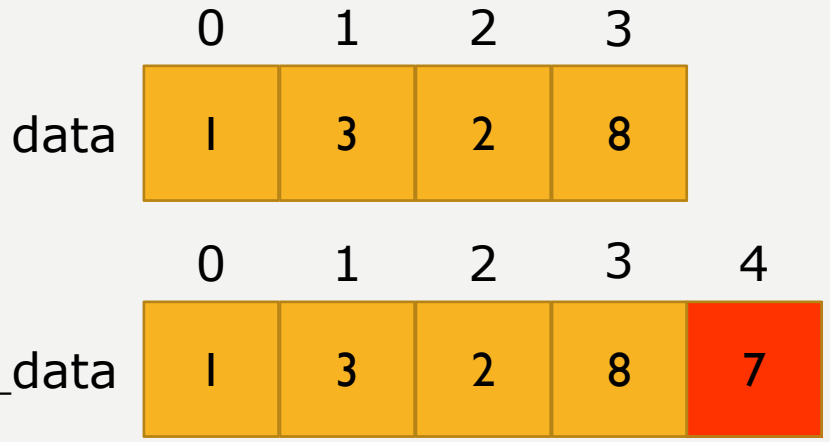

```
data.insert(2,5)
```




```
data.insert(2,5)
```



MODIFY A LIST VS. CREATE A LIST



- Which of the following can produce the result shown on the right?

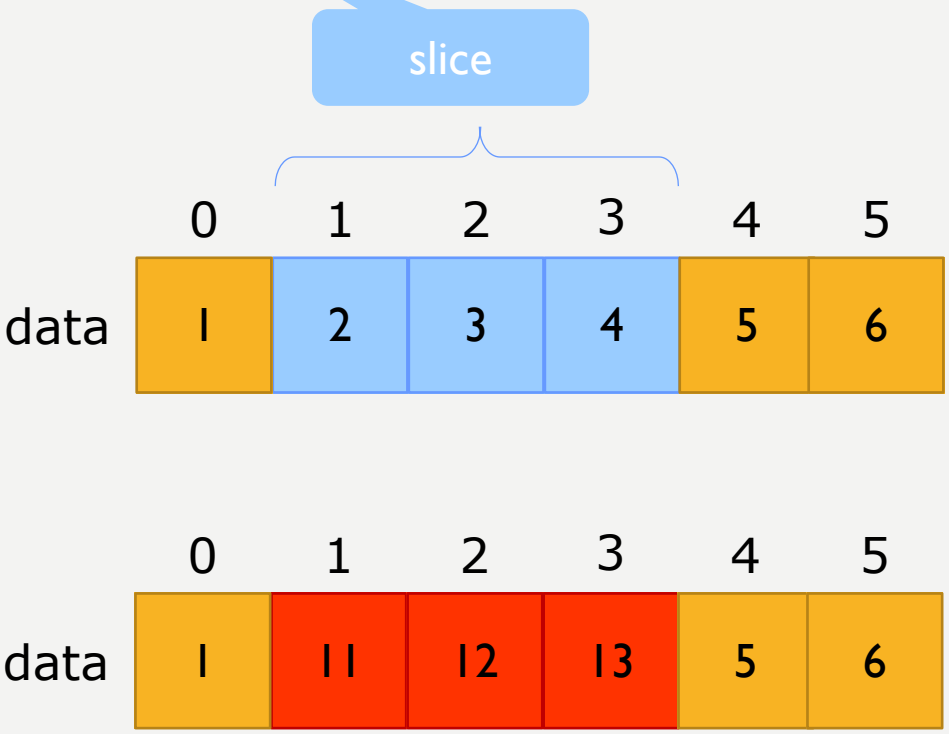
```
new_data=data.append(7)
```

```
new_data=data+[7]
```

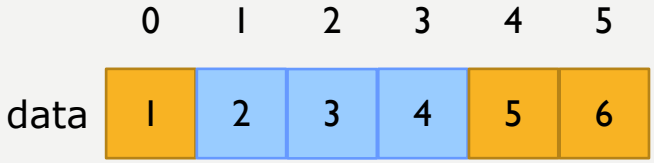
INSERTION (ADVANCE!!)

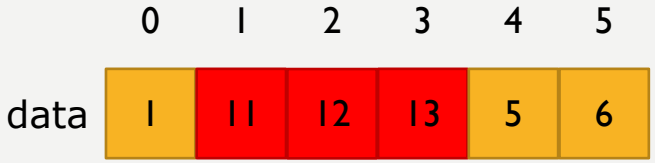
- Insert (a range of) item(s)

```
data=[1,2,3,4,5,6]

data[1:4] = [11, 12, 13]
```

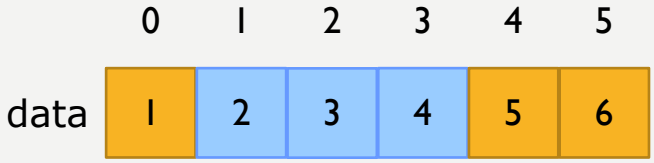


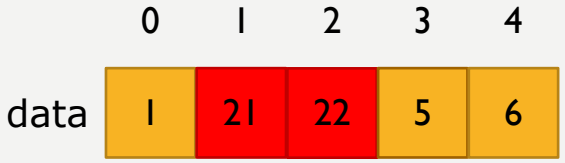
INSERTION (ADVANCE!!)



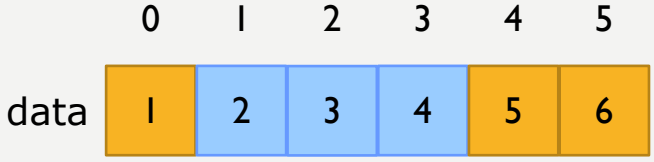


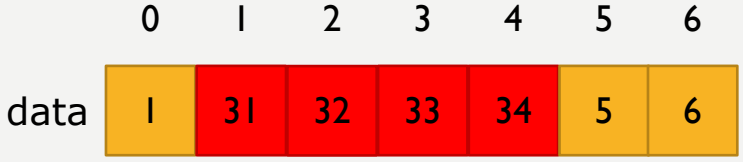

```
data[1:4] = [11, 12, 13]
```





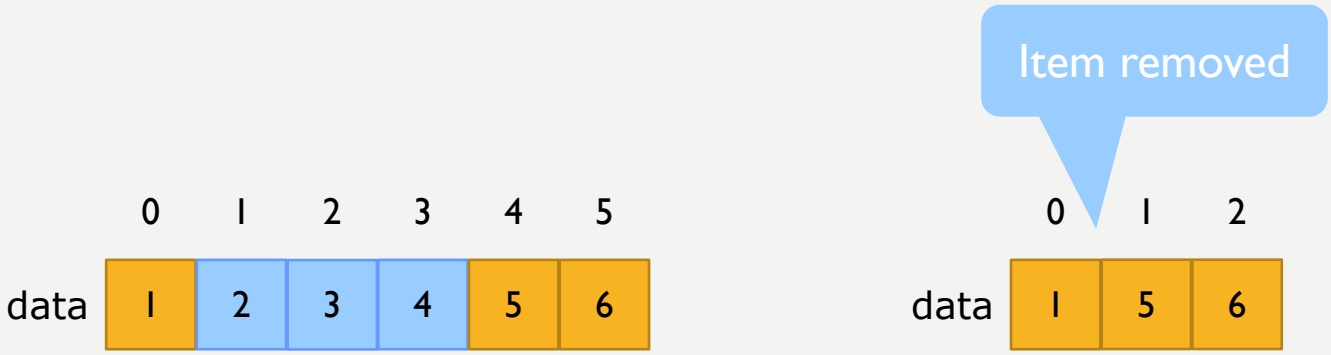

```
data[1:4] = [21, 22]
```





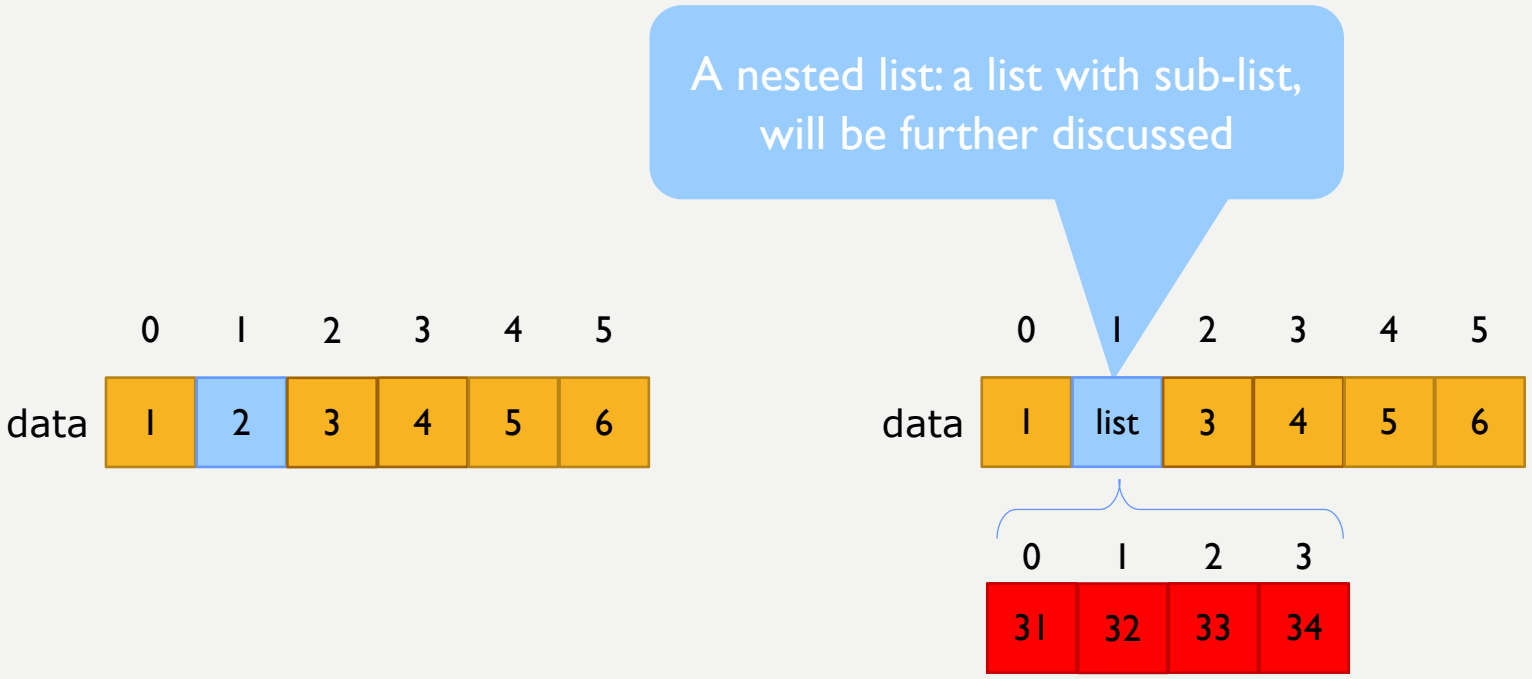

```
data[1:4] = [31, 32, 33, 34]
```

SLICE OPERATION ADVANCE!!




```
data[1:4] = []
```

LOOK OUT!!




```
data[1] = [31, 32, 33, 34]
```

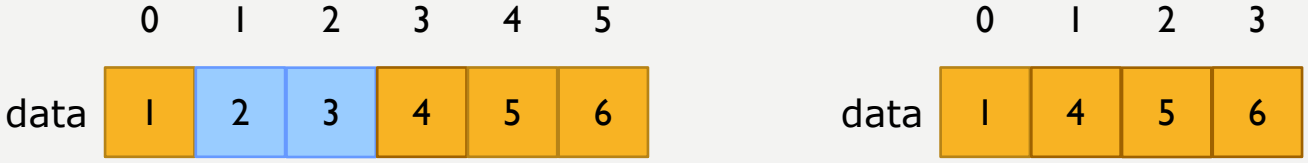
REMOVE AN ITEM FROM LIST

- Remove an item from a list

```
#remove the item at index
del list[index]


#remove a slice of items from a list
del list[start:end]
```

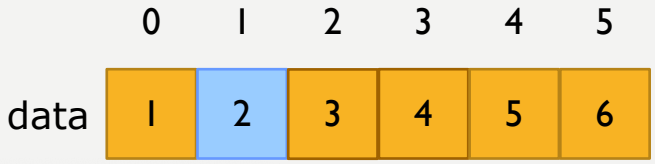
REMOVE AN ITEM FROM LIST


```
del data[1]
```




```
del data[1:3]
```

REMOVE AN ITEM FROM LIST




```
IDLE Shell 3.10.0
>>> data=[1,2,3,4,5,6]
...
>>> print(data)
...
    [1, 2, 3, 4, 5, 6]
>>> x=data.pop(1)
...
>>> data
...
    [1, 3, 4, 5, 6]
>>> x
...
    2
>>> y=data.pop()
...
>>> data
...
    [1, 3, 4, 5]
>>> y
...
    6
>>> data.remove(3)
...
>>> data
...
    [1, 4, 5]
>>>
Ln: 169  Col: 0
```

If you need the deleted item, use `pop`

If you don't provide an index, `pop` deletes and returns the last item

If you know the item (but not the index) you want to delete, use `remove`

SORT A LIST

- In Python, list can be easily ordered by calling the method `sort`.

```
labour=['David', 'Tom', 'John', 'Jimmy']

labour.sort()
print(labour)
```

```
['David', 'Jimmy', 'John', 'Tom']
```

```
labour.sort(reverse=True)
print(labour)
```

```
['Tom', 'John', 'Jimmy', 'David']
```

- If you want to create a new sorted list without modifying the original list, using the function `sorted`.

```
labour=['David', 'Tom', 'John', 'Jimmy']

labour_sorted=sorted(labour)
print(labour_sorted)
print(labour)
```

```
['David', 'Jimmy', 'John', 'Tom']
```

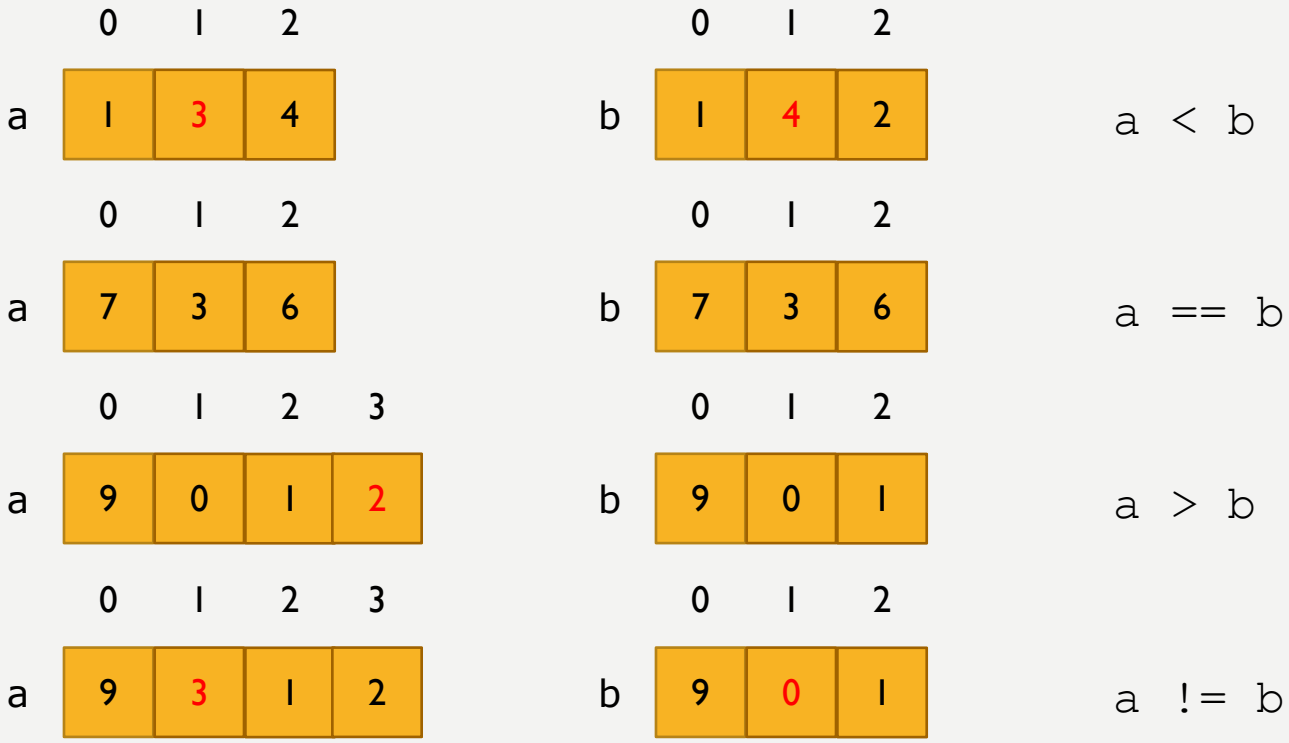
```
['David', 'Tom', 'John', 'Jimmy']
```

CONCATENATE LISTS

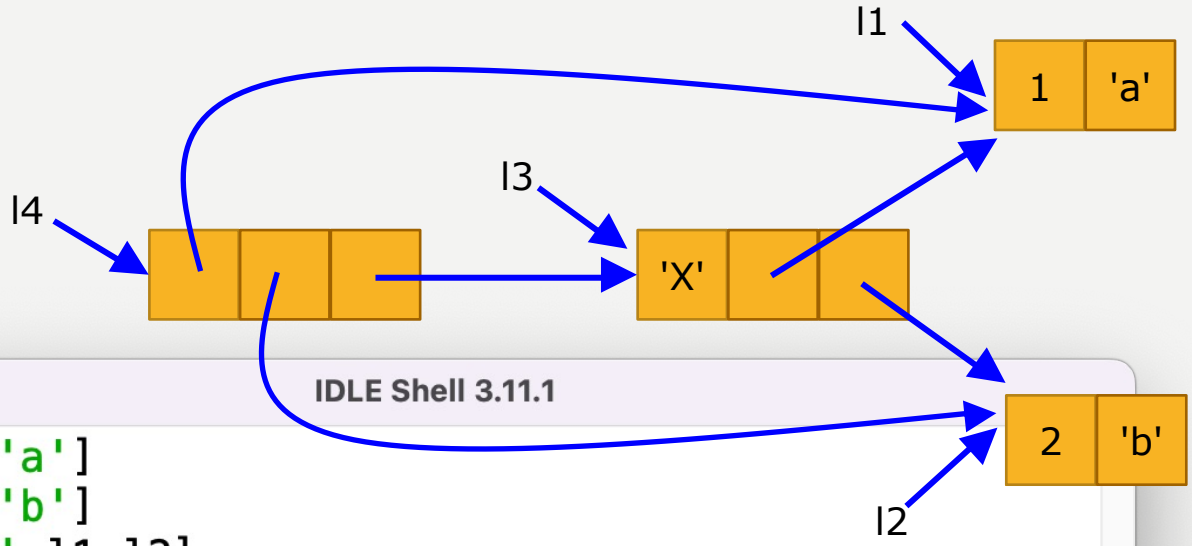
```
#concatenate two lists
l=[1, 2, 3]
m=[4, 5, 6]

n=l + m

o=l*2

p=l*2+m
```

LIST COMPARISON

- Standard comparisons (<, <=, >, >=, ==, !=, in, not in) also apply to list.
- Items in list are compared one by one.



NESTED LIST

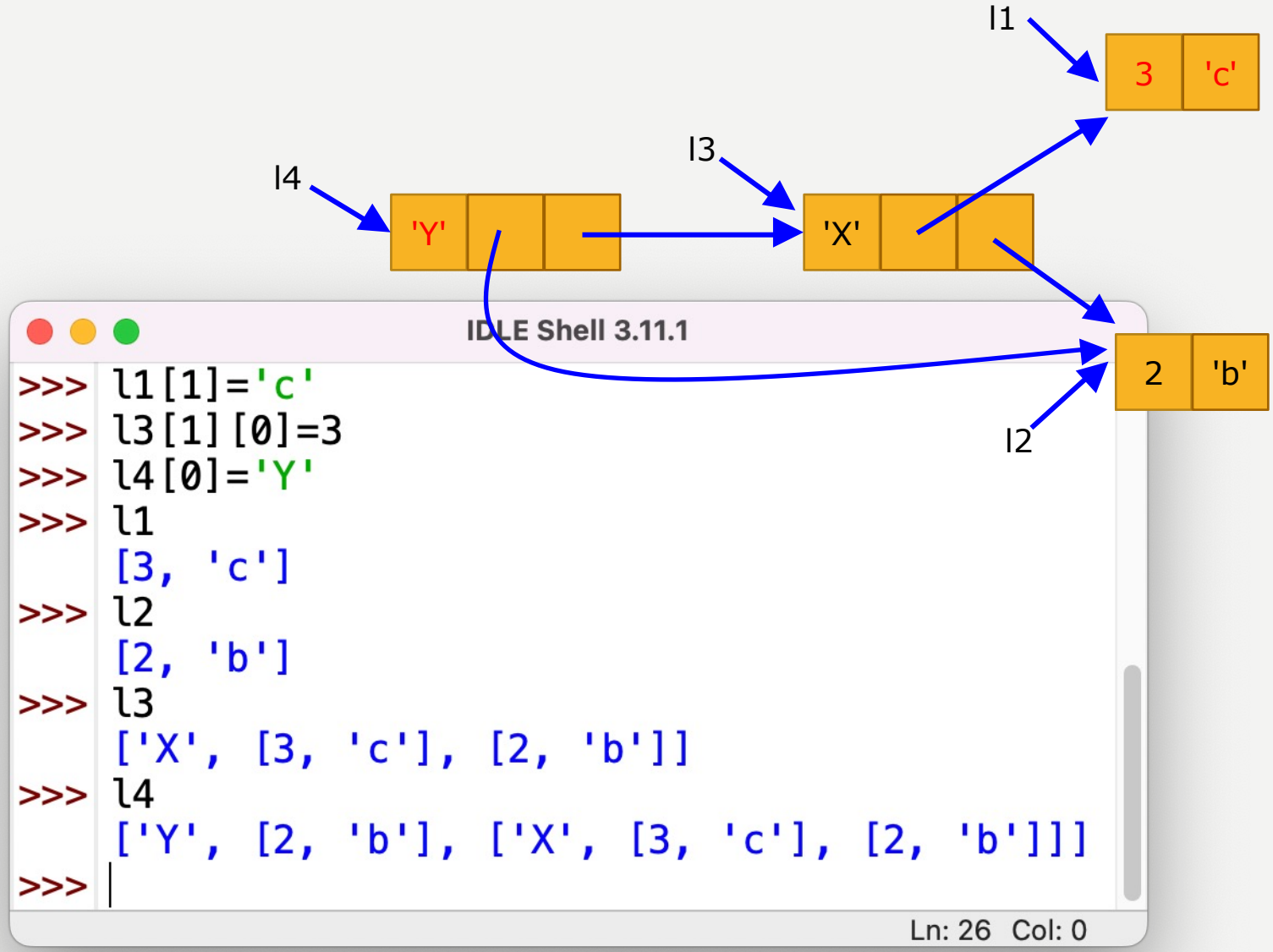


IDLE Shell 3.11.1

```
>>> l1=[1,'a']
>>> l2=[2,'b']
>>> l3=['X',l1,l2]
>>> l4=[l1,l2,l3]
>>> l1
[1, 'a']
>>> l2
[2, 'b']
>>> l3
['X', [1, 'a'], [2, 'b']]
>>> l4
[[1, 'a'], [2, 'b'], ['X', [1, 'a'], [2, 'b']]]
>>>
```

Ln: 138 Col: 0

NESTED LIST


l1
3
'c'
l3
l4
'Y'
'X'
2
'b'
l2
IDLE Shell 3.11.1
>>> l1[1]='c'
>>> l3[1][0]=3
>>> l4[0]='Y'
>>> l1
[3, 'c']
>>> l2
[2, 'b']
>>> l3
['X', [3, 'c'], [2, 'b']]
>>> l4
['Y', [2, 'b'], ['X', [3, 'c'], [2, 'b']]]
>>>
Ln: 26 Col: 0

PASS A LIST TO A FUNCTION

- List is **mutable**, it can be changed within a function
- Consider the following program that replaces all the negative number to positive

```
def toPositive(data):
    for i in range(len(data)):
        if data[i]<0:
            data[i]=-data[i]

d=[1, -1, 2, 3, -5, 3]
toPositive(d)
print(d)
```

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

ANOTHER WORKABLE VERSION?

- List is mutable, it can be changed within a function
- Consider the following program that replaces all the negative number to positive

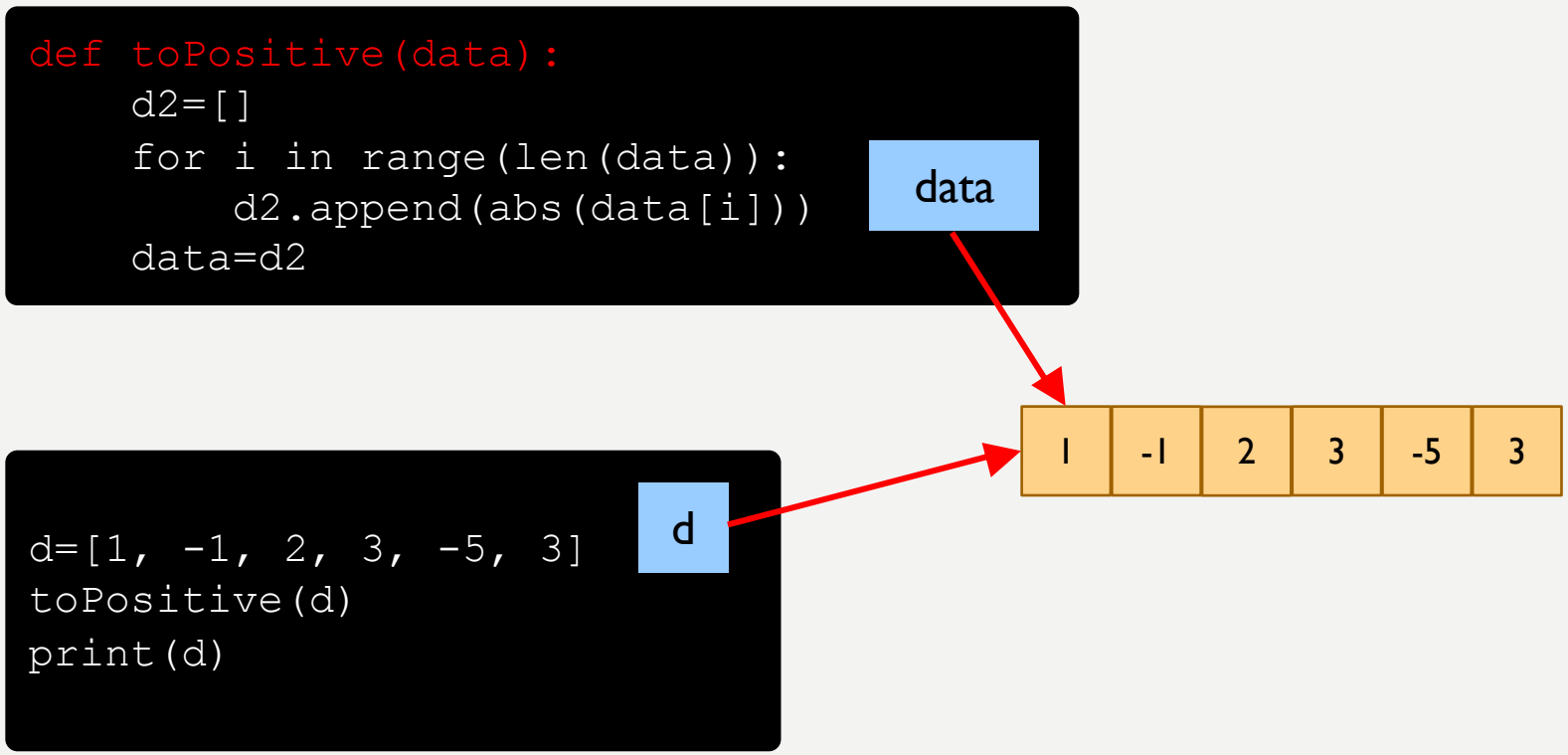
```
def toPositive(data):
    d2=[]
    for i in range(len(data)):
        d2.append(abs(data[i]))
    data=d2

d=[1, -1, 2, 3, -5, 3]
toPositive(d)
print(d)
```

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

BEHIND SCENE

```
def toPositive(data):
    d2=[]
    for i in range(len(data)):
        d2.append(abs(data[i]))
    data=d2
```

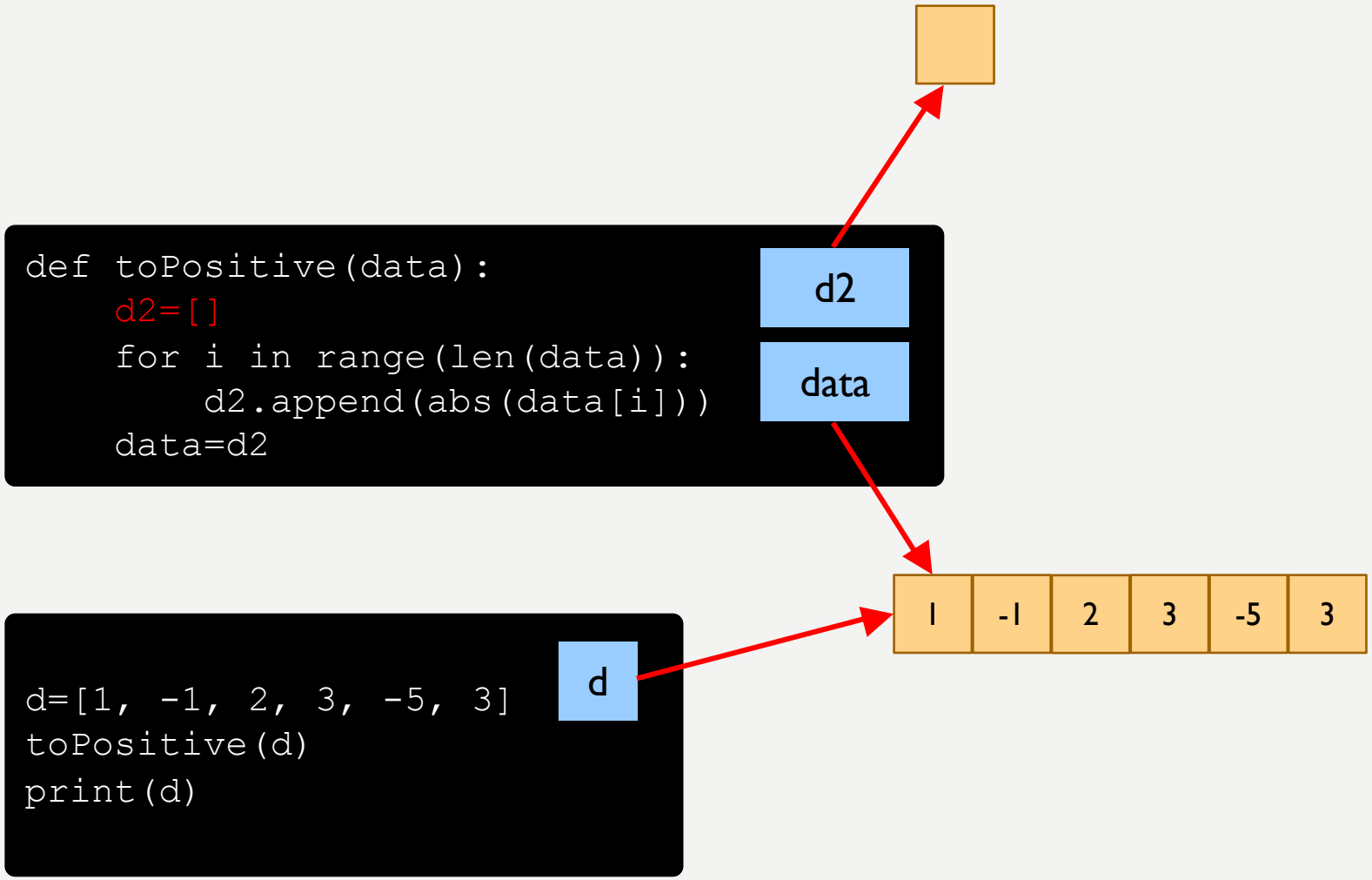
```
d=[1, -1, 2, 3, -5, 3]
toPositive(d)
print(d)
```



BEHIND SCENE

```
def toPositive(data):
    d2=[]
    for i in range(len(data)):
        d2.append(abs(data[i]))
    data=d2
```

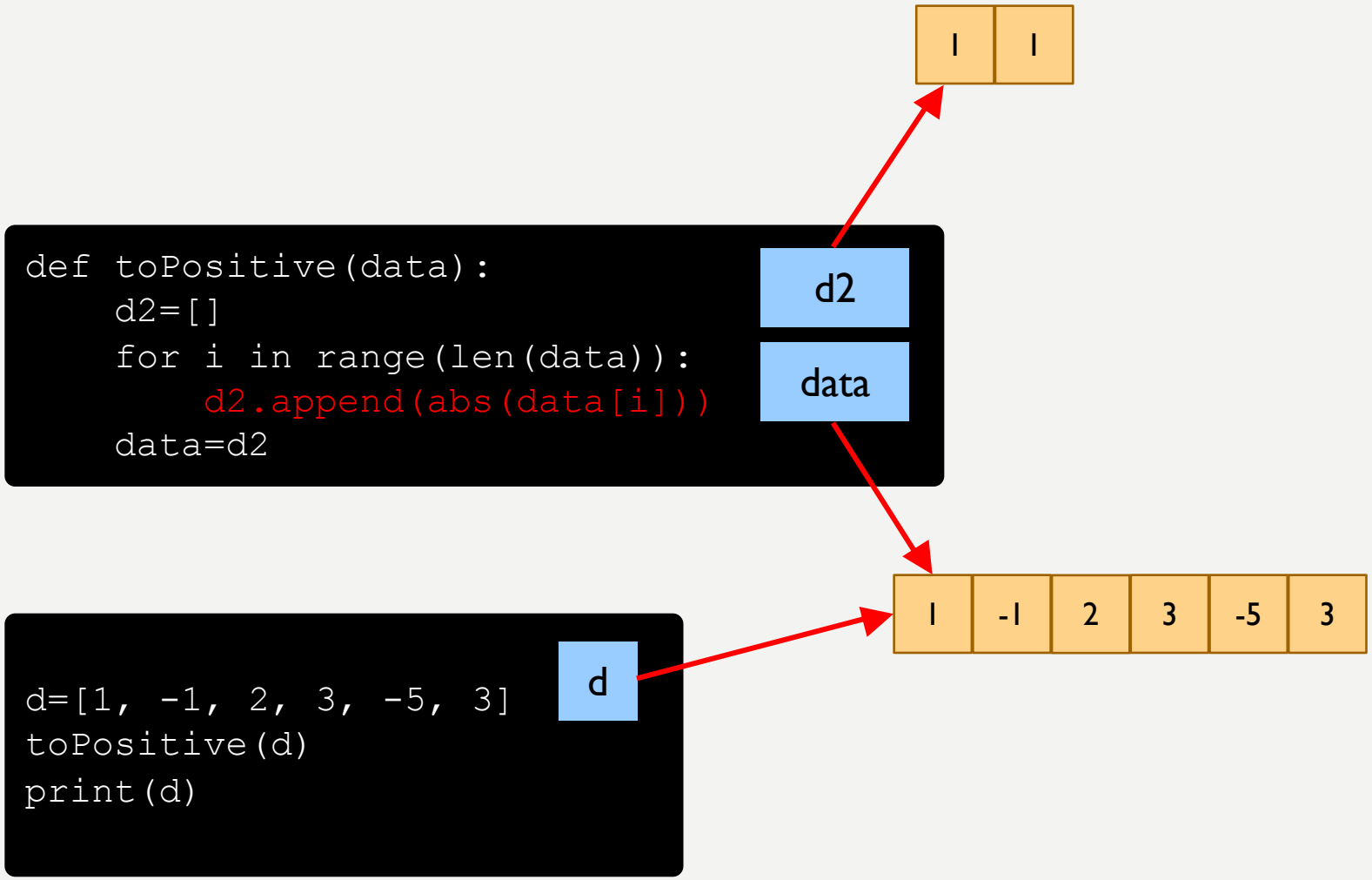
```
d=[1, -1, 2, 3, -5, 3]
toPositive(d)
print(d)
```



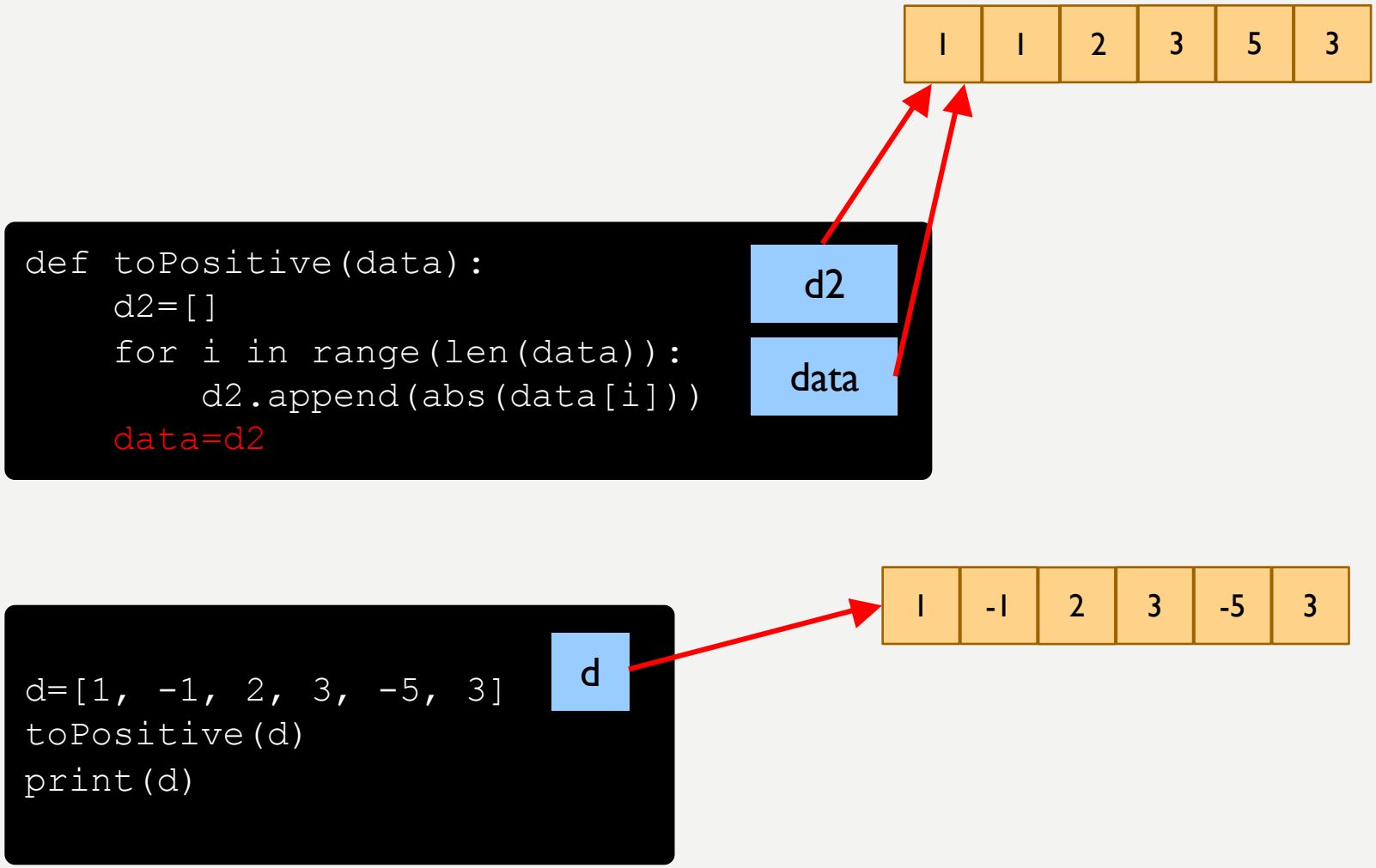
BEHIND SCENE

```
def toPositive(data):
    d2=[]
    for i in range(len(data)):
        d2.append(abs(data[i]))
    data=d2
```

```
d=[1, -1, 2, 3, -5, 3]
toPositive(d)
print(d)
```



BEHIND SCENE



SUMMARY

- Python supports array in the form of list.
- List's items can be accessed via index or iterated through via a `for` loop.
- List can store data of different types.
- List can be dynamically grown and shrunk in size.
- Items can be inserted into a list via `list.append()` and `list.insert()`.
- Items can be deleted from the list via `del`, `list.pop()` and `list.remove()`.
- Slicing can both add items to and delete items from list.
- List can be changed within a function.