LISTS

LISTS – TODAY'S TOPICS

What are lists

List Characteristics

Accessing items in lists and iterating over Lists

Creating , Concatenating, Multiplying

Processing elements in lists

Using methods associated with list

WHAT ARE LISTS?

LISTS

A list is built-in type in Python.

It is used to store multiple values in a single variable.

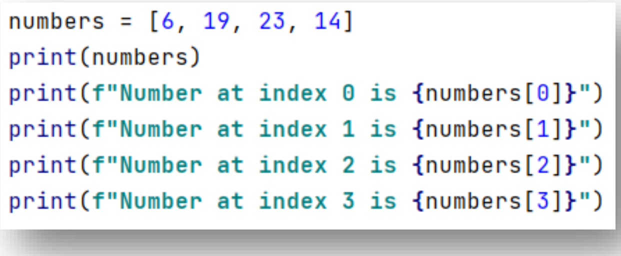
Lists are created using **[]**

```
a_list = [5, 7, 3, 9]
colours = ['red', 'green', 'blue']
```

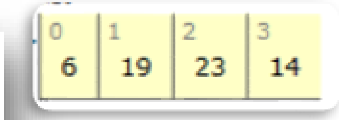
LISTS

Use [ ] to access parts of the list.

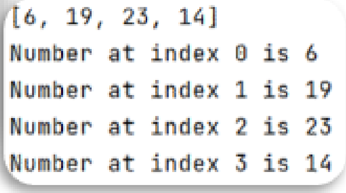
The first item is at **index 0**.



```
numbers = [6, 19, 23, 14]
print(numbers)
print(f"Number at index 0 is {numbers[0]}")
print(f"Number at index 1 is {numbers[1]}")
print(f"Number at index 2 is {numbers[2]}")
print(f"Number at index 3 is {numbers[3]}")
```



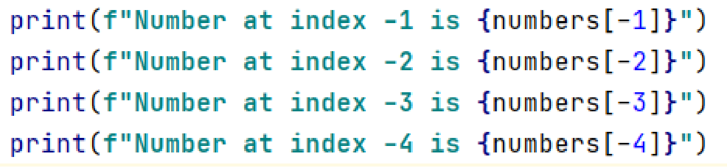
0	1	2	3
6	19	23	14



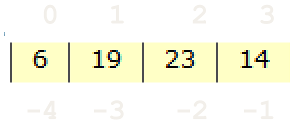
```
[6, 19, 23, 14]
Number at index 0 is 6
Number at index 1 is 19
Number at index 2 is 23
Number at index 3 is 14
```

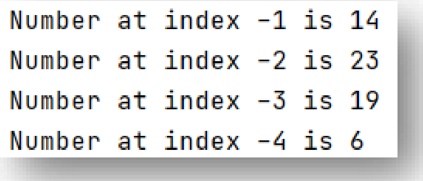
NEGATIVE INDEX

Negative numbers can be used to access items from the **end** of the list.



```
print(f"Number at index -1 is {numbers[-1]}")
print(f"Number at index -2 is {numbers[-2]}")
print(f"Number at index -3 is {numbers[-3]}")
print(f"Number at index -4 is {numbers[-4]}")
```

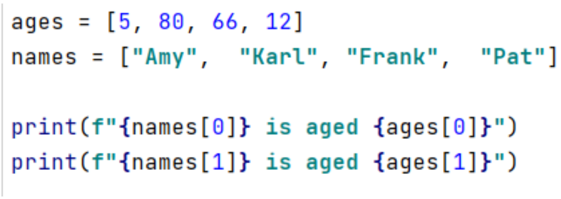




```
Number at index -1 is 14
Number at index -2 is 23
Number at index -3 is 19
Number at index -4 is 6
```

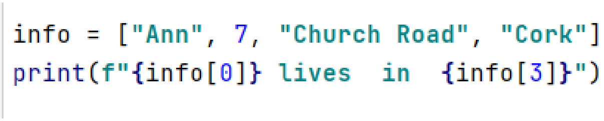
LISTS CHARACTERISTICS (LIKE TUPLES)

A list can be of all the same type - **homogenous** list



```
ages = [5, 80, 66, 12]
names = ["Amy",  "Karl", "Frank",  "Pat"]

print(f"{names[0]} is aged {ages[0]}")
print(f"{names[1]} is aged {ages[1]}")
```

A list of different types of items - **heterogeneous** list

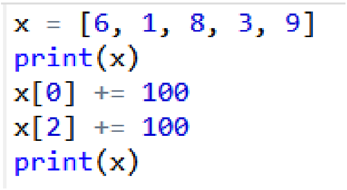


```
info = ["Ann", 7, "Church Road", "Cork"]
print(f"{info[0]} lives  in  {info[3]}")
```

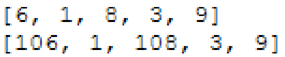
But Python documentation says that this is *not* what lists are for and you should stick with homogenous lists.

LISTS CHARACTERISTICS (NOT LIKE TUPLES)

Any of the items in a list may be changed - lists are **mutable**

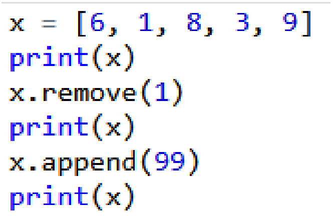


```
x = [6, 1, 8, 3, 9]
print(x)
x[0] += 100
x[2] += 100
print(x)
```

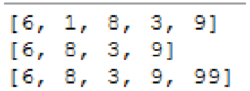


```
[6, 1, 8, 3, 9]
[106, 1, 108, 3, 9]
```

Lists are **dynamic** - we can add an remove items from a list using functions `append` and `remove`



```
x = [6, 1, 8, 3, 9]
print(x)
x.remove(1)
print(x)
x.append(99)
print(x)
```

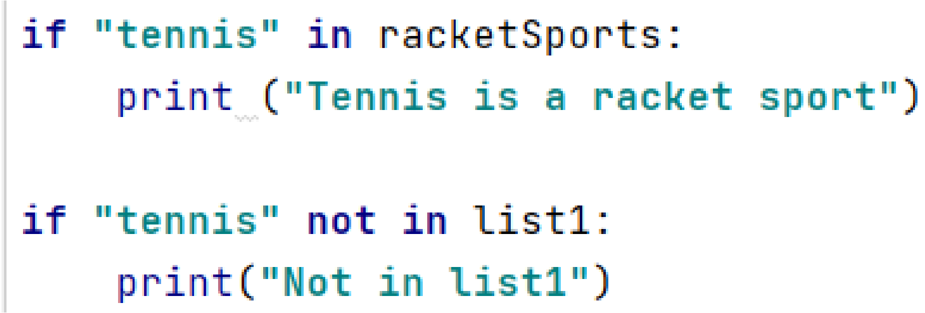


```
[6, 1, 8, 3, 9]
[6, 8, 3, 9]
[6, 8, 3, 9, 99]
```

IN (JUST LIKE TUPLES)

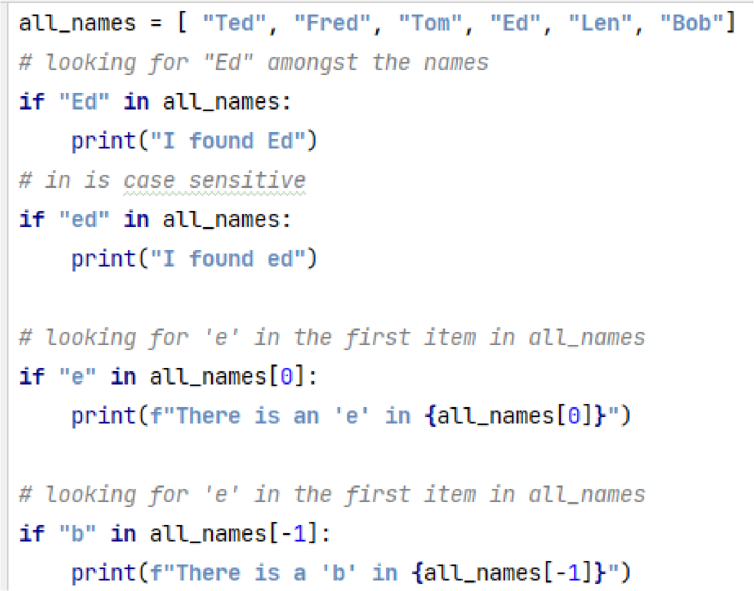
can be used to determine if an item is contained in a list.

- returns `True` if the item is in the list.
- returns `False` if the item is not in the list

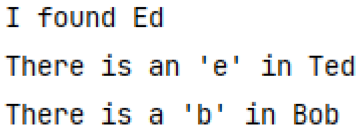


```
if "tennis" in racketSports:
    print ("Tennis is a racket sport")

if "tennis" not in list1:
    print("Not in list1")
```

IN (JUST LIKE TUPLES)



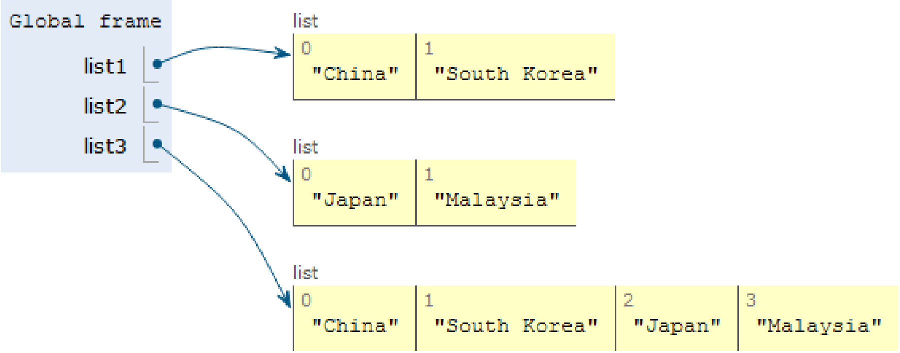
```python
all_names = [ "Ted", "Fred", "Tom", "Ed", "Len", "Bob"]
# looking for "Ed" amongst the names
if "Ed" in all_names:
    print("I found Ed")
# in is case sensitive
if "ed" in all_names:
    print("I found ed")

# looking for 'e' in the first item in all_names
if "e" in all_names[0]:
    print(f"There is an 'e' in {all_names[0]}")

# looking for 'e' in the first item in all_names
if "b" in all_names[-1]:
    print(f"There is a 'b' in {all_names[-1]}")
```



```
I found Ed
There is an 'e' in Ted
There is a 'b' in Bob
```

ADDING LISTS MAKES A NEW BIGGER LIST

```
list1 = ["China", "South Korea"]
list2 = ["Japan", "Malaysia"]
list3 = list1 + list2
print(list3)
```



Global frame

list1

list2

list3

list

0	1
"China"	"South Korea"

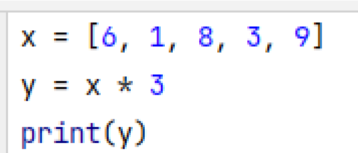
list

0	1
"Japan"	"Malaysia"

list

0	1	2	3
"China"	"South Korea"	"Japan"	"Malaysia"

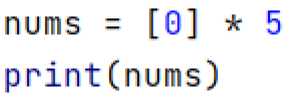
MULTIPLYING LISTS BY INTEGER MAKES A NEW BIGGER LIST



```
x = [6, 1, 8, 3, 9]
y = x * 3
print(y)
```

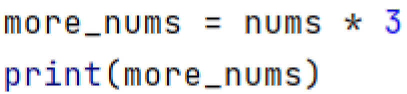
```
6, 1, 8, 3, 9, 6, 1, 8, 3, 9, 6, 1, 8, 3, 9]
```

USE CASE FOR MULTIPLYING BY INTEGER



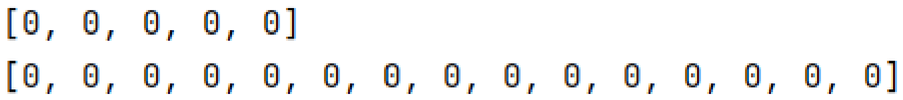
```
nums = [0] * 5
print(nums)
```

Create a list called nums containing 5 0's.



```
more_nums = nums * 3
print(more_nums)
```

Create a list of 15 0's



```
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

LOOP THROUGH A LIST

FOR WITH NEW VARIABLE

The for construct

for **variable** in a_**list**

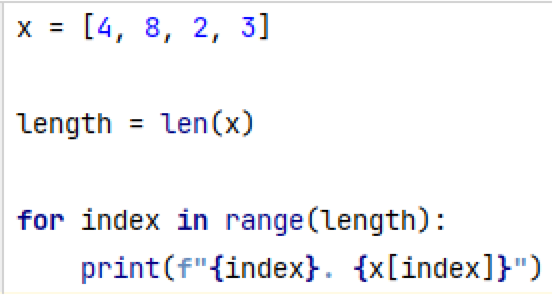
is the easiest way to look at each element in a list (or other collection).

```
names = ["Donald", "Minnie", "Daisy"]
for name in names:
    print(name)
```

```
Donald
Minnie
Daisy
```

FOR WITH LEN()/RANGE()

`len()` returns the number of values in a list – its length



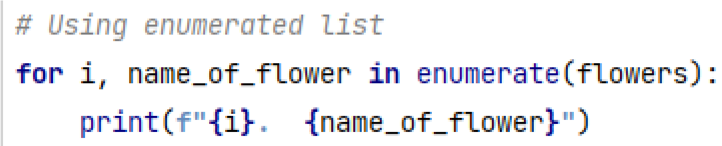
```python
x = [4, 8, 2, 3]

length = len(x)

for index in range(length):
    print(f"{index}. {x[index]}")
```

```
0. 4
1. 8
2. 2
3. 3
```

ENUMERATE THROUGH A LIST

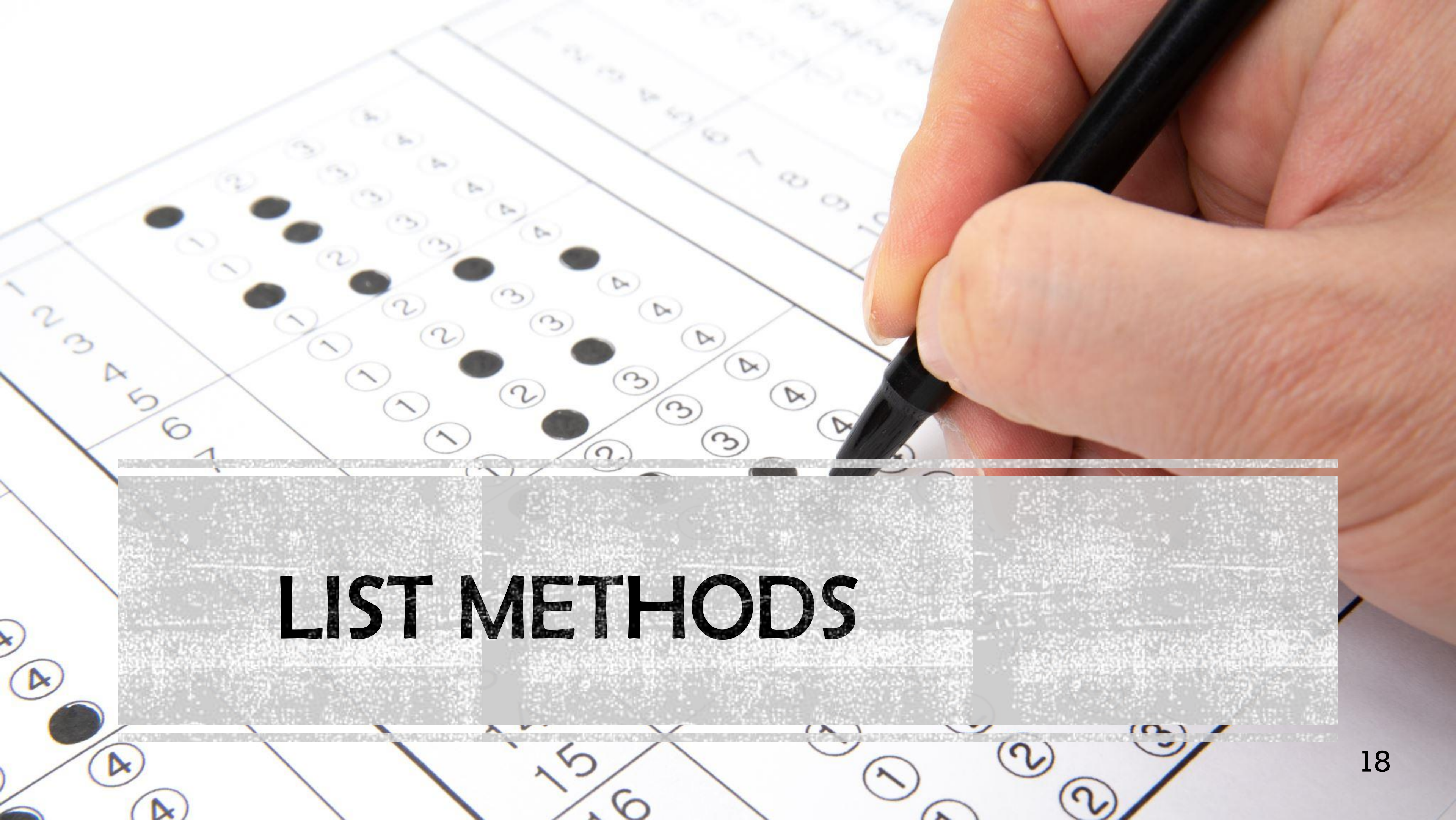
The enumerate function returns (unpacks) two loop variables in turn:

- the index of the object
- the value of the object



```python
# Using enumerated list
for i, name_of_flower in enumerate(flowers):
    print(f"{i}.  {name_of_flower}")
```

```
0.   Pansy
1.   Bluebell
2.   Crocus
```

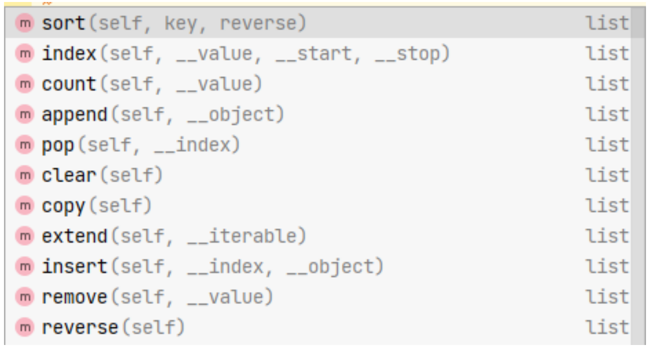


LIST METHODS

METHODS

A list has in-built functions accessed using the dot operator (.)

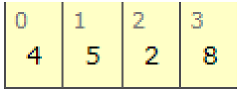
Functions accessed using dot operators are called ***methods***



```
m sort(self, key, reverse)                      list
m index(self, __value, __start, __stop)         list
m count(self, __value)                          list
m append(self, __object)                        list
m pop(self, __index)                            list
m clear(self)                                   list
m copy(self)                                    list
m extend(self, __iterable)                      list
m insert(self, __index, __object)               list
m remove(self, __value)                         list
m reverse(self)                                 list
```

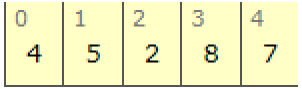
MAKE THE LIST BIGGER

x = [4,5,2,8]



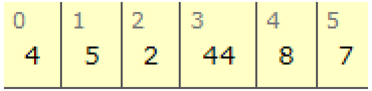
0	1	2	3
4	5	2	8

x.append(7)



0	1	2	3	4
4	5	2	8	7

x.insert(3, 44)



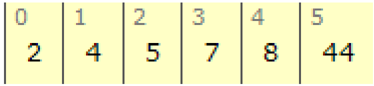
0	1	2	3	4	5
4	5	2	44	8	7

insert 44 at index 3

list.append(elem) -- adds a single item to the end of the list.
list.insert(index, elem) -- inserts the element at the given index, shifting elements to the right.

SORT THIS LIST

x.sort()



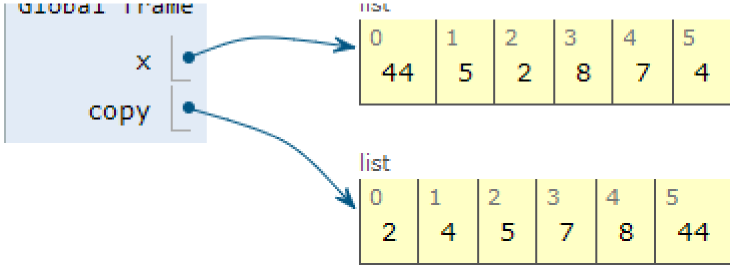
0	1	2	3	4	5
2	4	5	7	8	44

x.sort() -- sorts the list in place ie. does not return it.

x.sort(reverse=True) reverses the list

SORTED() FUNCTION

```
x = [44, 5, 2, 8, 7, 4]
copy = sorted(x)
```



0	1	2	3	4	5
44	5	2	8	7	4

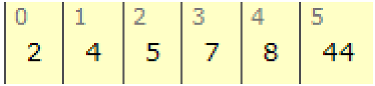
0	1	2	3	4	5
2	4	5	7	8	44

You will often see sorted() function in Python.
sorted() returns a **new** sorted list, leaving the original list unaffected.

list.sort() sorts the list **in-place**, mutating the list indices, and returns None

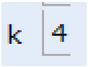
list.sort() is faster because no copy is created.
Use sorted() if you need to keep a copy of the original, otherwise use list.sort()

SEARCHING FOR ELEMENT



0	1	2	3	4	5
2	4	5	7	8	44

```
k =  x.index(8)


k = x.index(99)
```



k	4

```
ValueError: 99 is not in list
```

list.index(elem) -- searches for elem from the start of the list and returns its index.

Throws a ValueError if the element does not appear.
Use "in" to verify item in list before using index.

INDEX() AND IN

```
x = [2,4,5,7,8,44]
seeking = int(input("Which number are you looking for? "))
if seeking in x:
    print(f"{seeking} is at index {x.index(seeking)}")
else:
    print(f"{seeking} not in the list")
```

REMOVE()

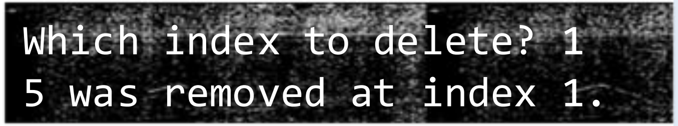
list.remove(elem) -- removes the first instance of elem in-place. Throws ValueError if not present so use in first

```
x = [4,5,2,8,7,3]
to_remove = int(input("Which number to delete? "))
if to_remove in x:
    x.remove(to_remove)
else:
    print(f"{to_remove} not in the list")
```

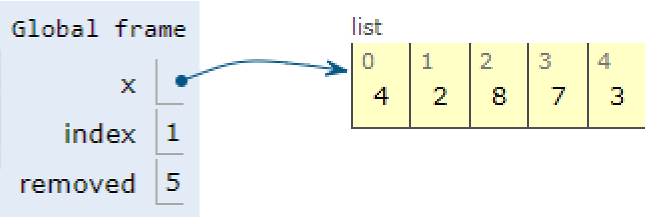
POP()

removed = list.pop(index)
Remove and return item at index (default last).
Raises IndexError if list is empty, or index is out of range.

```
x = [4,5,2,8,7,3]
index = int(input("Which index to delete? "))
removed = x.pop(index)
print(f"{removed} was removed at index {index}.")
```



```
Which index to delete? 1
5 was removed at index 1.
```



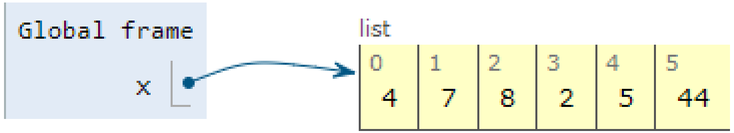
POP()

```
x = [5,6,4,7,2,8]
index = int(input("Which index to delete? "))

try:
    removed = x.pop(index)
    print(f"{removed} was removed at index {index}.")
except IndexError:
    print(f"{index} index was out of range. ")
```

REVERSE()

list.reverse() -- reverses the list in place (does not return it)

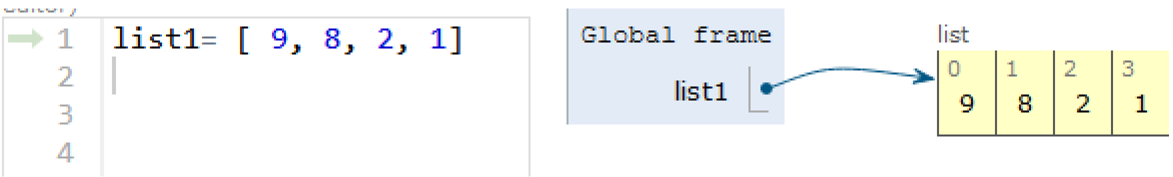
```
x = [44, 5, 2, 8, 7, 4]
x.reverse()
```



Common misunderstand – this method does not rearrange the list in reverse *order* it just reverses the list.

If you need the list in reverse order, first sort it then reverse it.

ADDRESS OF LIST



```
list1= [ 9, 8, 2, 1]
```

- list1 is a variable not a list.
- list1 holds the address of the first item in the list.
- list1 is a reference or pointer to the list.

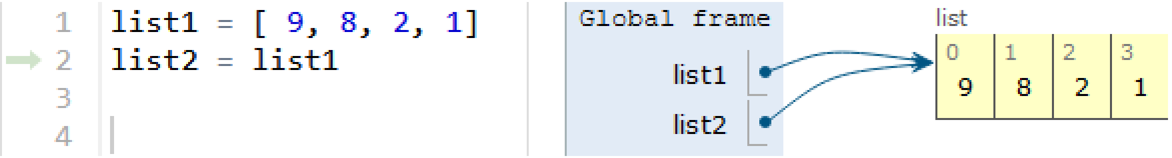
TWO LISTS – SAME ADDRESS

Using = on lists does not make a copy.

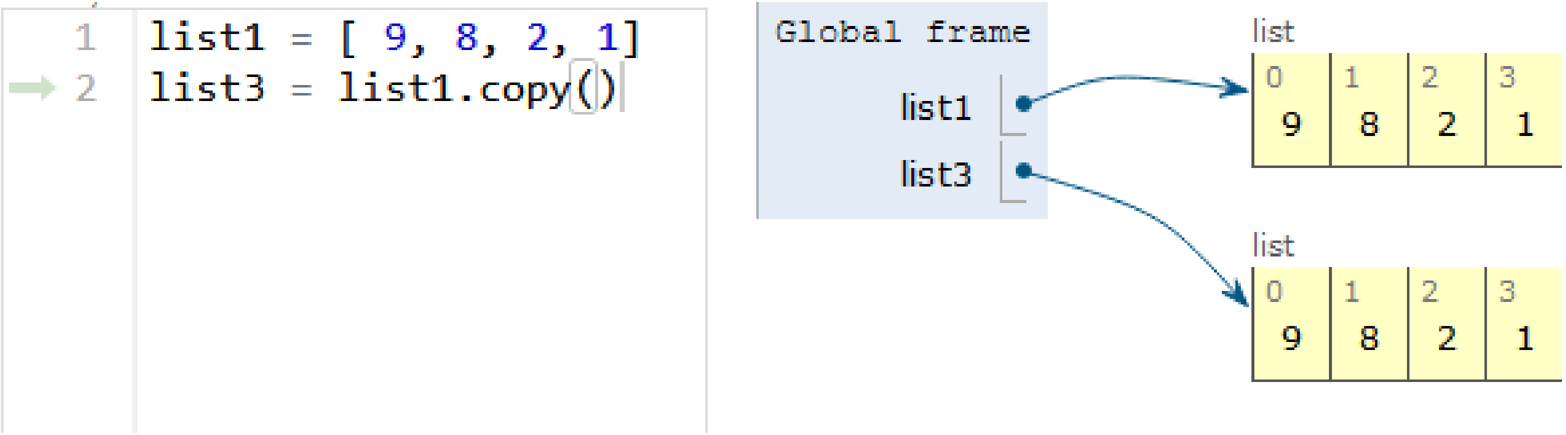
Instead, both variables point to the same list.

```
list2 = list1
```

- list2 gets the address contained in list1.
- list2 points to the same list as list1.



```
list1 = [ 9, 8, 2, 1]
list2 = list1
```



```
list1 = [ 9, 8, 2, 1]
list3 = list1.copy()
```

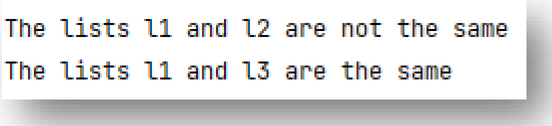
COPY()

If you must make a copy of a list use copy()

==

== compares items in a list one at a time

```
l1 = [1, 2, 3, 4, 5]
l2 = [5, 1, 3, 2, 4]  # same values as l1, different order
l3 = [1, 2, 3, 4, 5]  # same values as l1, same order

if l1 == l2:
    print("The lists l1 and l2 are identical")
else:
    print("The lists l1 and l2 are not identical")

if l1 == l3:
    print("The lists l1 and l3 are identical")
else:
    print("The lists l1 and l3 are not identical")
```



```
The lists l1 and l2 are not the same
The lists l1 and l3 are the same
```

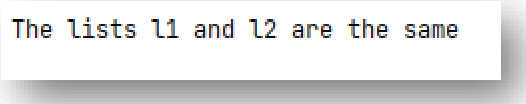
IS VS ==

```python
l1 = [1, 2, 3, 4, 5]
l2 = [1, 2, 3, 4, 5]  # same values as l1, same order
l3 = l1  # same values as l1, same order

if l1 is l3:
    print("l1 and l2 are the same list.")

if l1 is l2:
    print("l1 and l3 are the same list.")

if l1 == l2:
    print("The lists l1 and l2 are identical.")
else:
    print("The lists l1 and l2 are not identical.")

if l1 == l3:
    print("The lists l1 and l3 are identical.")
else:
    print("The lists l1 and l3 are not identical.")
```

== AND SORTED()

If you don't care about the order, you just want to compare the contents then create a sorted copy of each and compare those.

```
l1 = [1, 2, 3, 4, 5]
l2 = [5, 1, 3, 2, 4]  # same values as l1, different order
l1_sorted = sorted(l1)
l2_sorted = sorted(l2)

if l1_sorted == l2_sorted:
    print("The lists l1 and l2 are the same")
else:
    print("The lists l1 and l2 are not the same")
```



```
The lists l1 and l2 are the same
```

PYTHON GIVES US FUNCTIONS THAT OPERATE *ON* LISTS

These Python functions receive a list as an argument and process it.

THE MOST USEFUL ARE...

- min()
- len()
- max()
- sum()

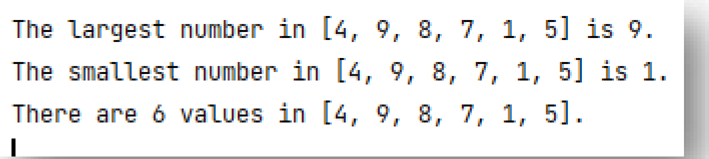
```
x = [5,6,4,7,2,8]
average = sum(x) / len(x)
print(f"The average of the numbers is {average:.2f}.")
```

AVERAGE

sum() is a function that receives a list of numbers, adds up the numbers in that list and returns the total.

MAX AND MIN

```
nums = [4, 9, 8, 7, 1, 5]

largest = max(nums)
smallest = min(nums)
number_of_numbers = len(nums)

print(f"The largest number in {nums} is {largest}.")
print(f"The smallest number in {nums} is {smallest}.")
print(f"There are {number_of_numbers} values in {nums}.")
```



```
The largest number in [4, 9, 8, 7, 1, 5] is 9.
The smallest number in [4, 9, 8, 7, 1, 5] is 1.
There are 6 values in [4, 9, 8, 7, 1, 5].
```

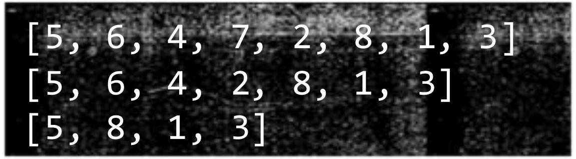
DEL STATEMENT

del is a statement, not a method or function.

No brackets or dots are needed when using del.

del removes elements at a particular index/indices.

```
x = [5, 6, 4, 7, 2, 8, 1, 3]
print(x)
del x[3]
print(x)
del x[1:4]
print(x)
```



```
[5, 6, 4, 7, 2, 8, 1, 3]
[5, 6, 4, 2, 8, 1, 3]
[5, 8, 1, 3]
```

DIY

- Create a list of five numbers
- Print the first item and last time in the list
- Use a loop to multiple each item in the list by 11.
- Use a for loop to print all the items in the list
- Use an enumerated for loop to print and number all the items

THANKS!

Any questions?