Lists

Lists [ ]

What is a python list?

1. An ordered collection
2. That is **resizable**
3. And contain elements of different types

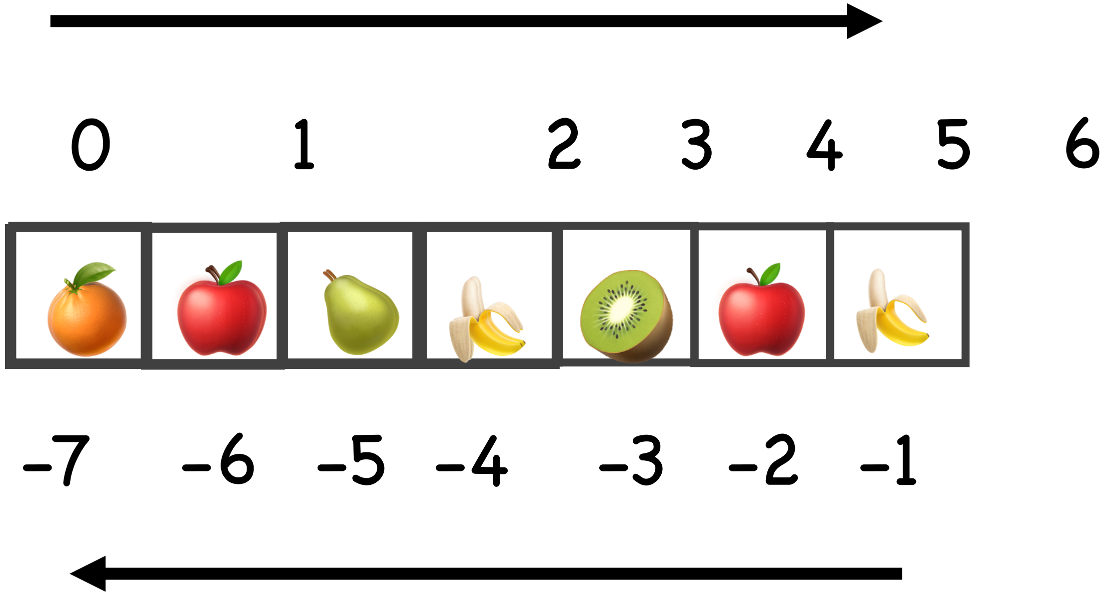
```
# Create a list
num_list = [3, 1, 2]

# A list can have all types
mix_list = ['hello',1,True]
```

How to access elements of a list ?

List Indexing

How do we access elements of a list ?

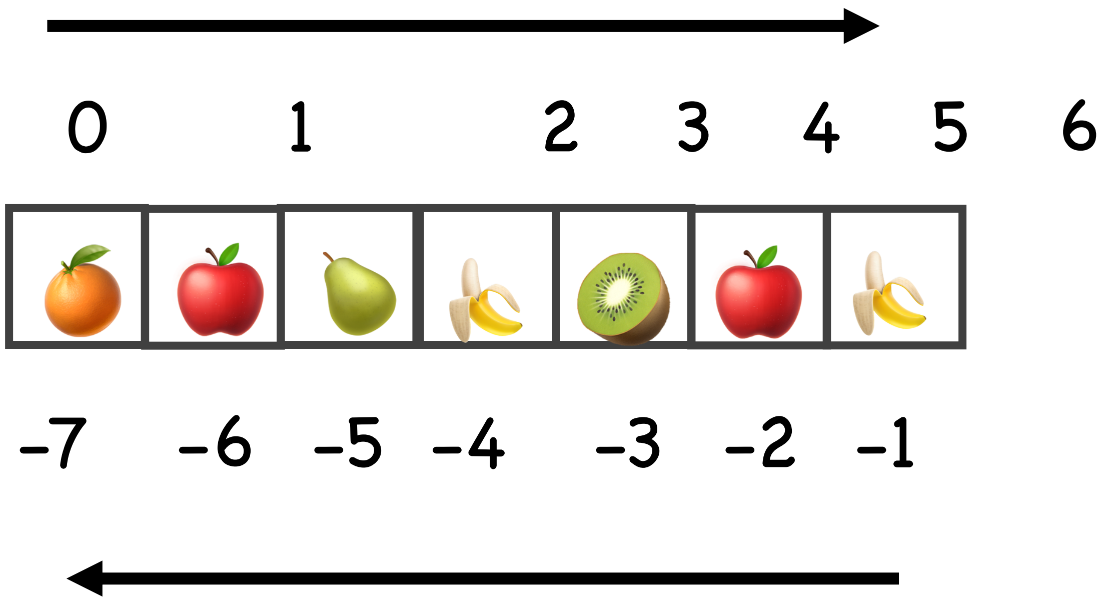
```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple',
'pear', 'banana', 'kiwi', 'apple',
'banana']
```



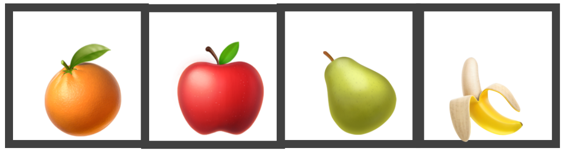
List[start:stop:(Optional)step]

List Indexing

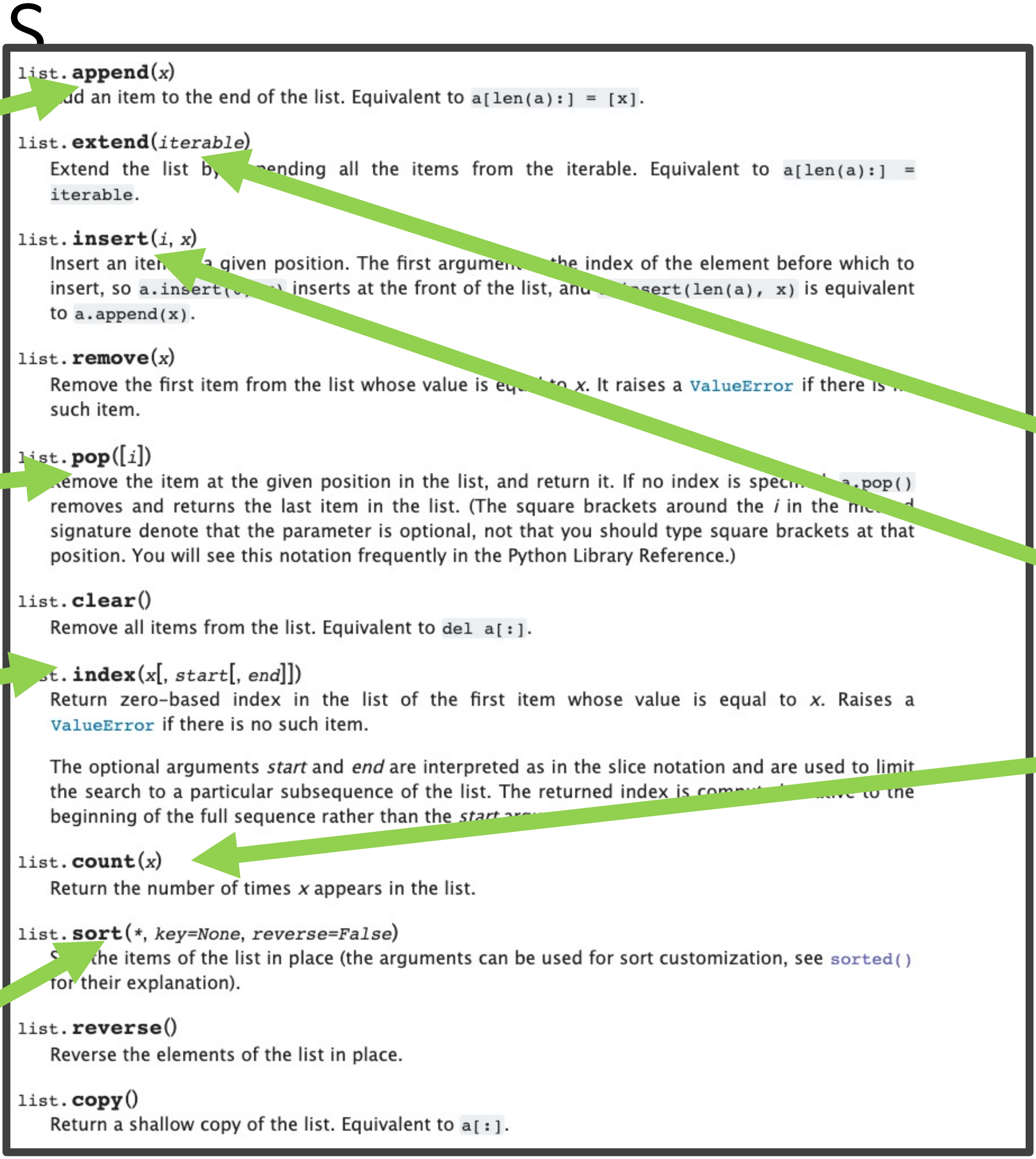
```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple',
'pear', 'banana', 'kiwi', 'apple',
'banana']
```



fruits[0:4] =



List Methods



`list.`**`append`**`(x)`
Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

`list.`**`extend`**`(iterable)`
Extend the list by appending all the items from the iterable. Equivalent to `a[len(a):] = iterable`.

`list.`**`insert`**`(i, x)`
Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

`list.`**`remove`**`(x)`
Remove the first item from the list whose value is equal to *x*. It raises a `ValueError` if there is no such item.

`list.`**`pop`**`([i])`
Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

`list.`**`clear`**`()`
Remove all items from the list. Equivalent to `del a[:]`.

`list.`**`index`**`(x[, start[, end]])`
Return zero-based index in the list of the first item whose value is equal to *x*. Raises a `ValueError` if there is no such item.

The optional arguments *start* and *end* are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the *start* argument.

`list.`**`count`**`(x)`
Return the number of times *x* appears in the list.

`list.`**`sort`**`(*, key=None, reverse=False)`
Sort the items of the list in place (the arguments can be used for sort customization, see `sorted()` for their explanation).

`list.`**`reverse`**`()`
Reverse the elements of the list in place.

`list.`**`copy`**`()`
Return a shallow copy of the list. Equivalent to `a[:]`.

add to a list

extend with another list

insert an item in list

remove from
the list

get location
of an item

count an item

sort the list

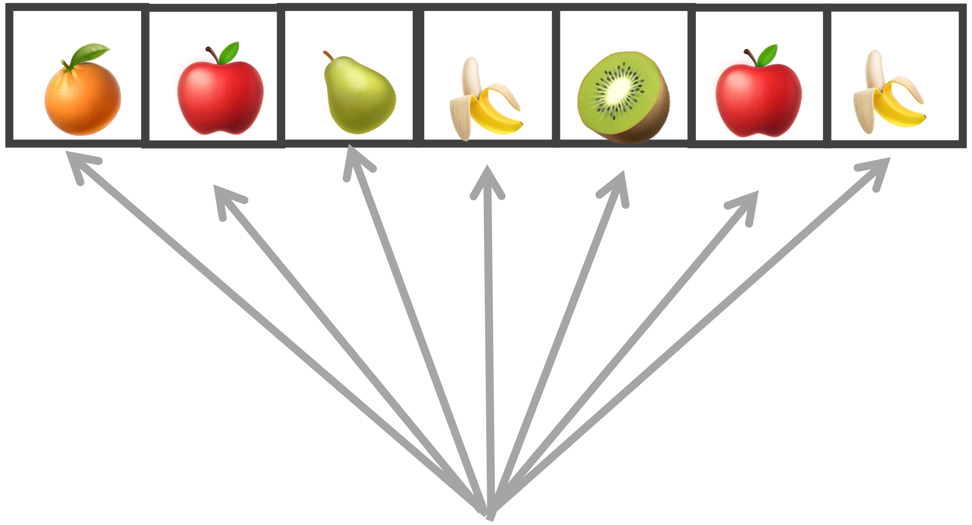
List Methods

List methods

```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

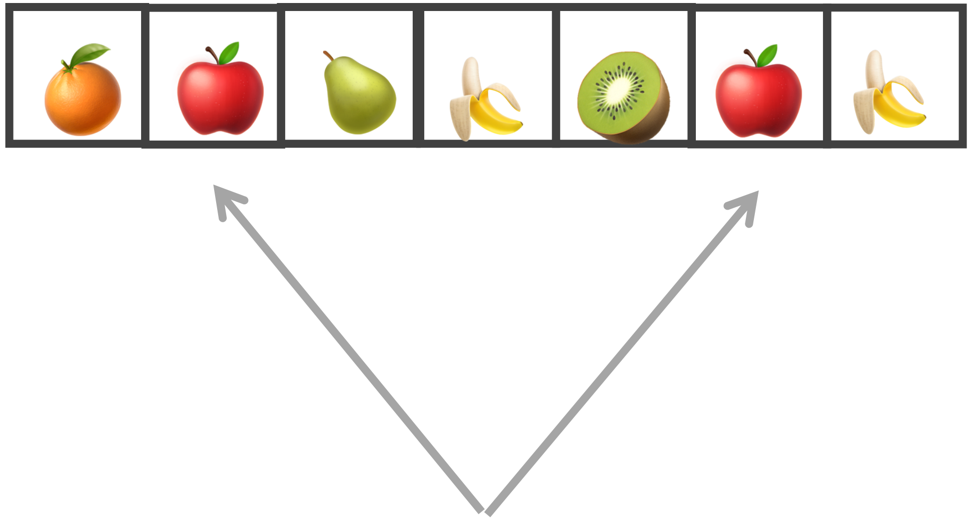
List methods

```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']


>>> len(fruits)
7
```



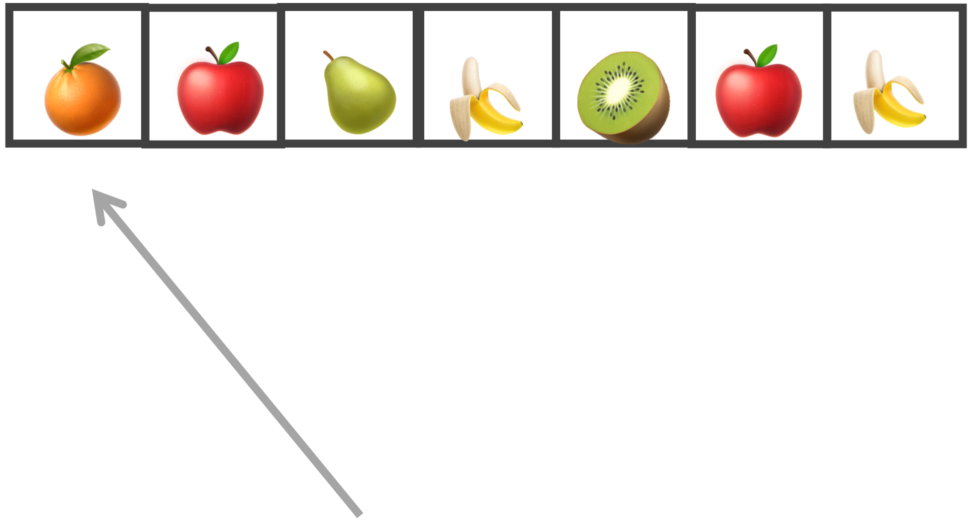
Count()

```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']

>>> fruits.count('apple')
2
```



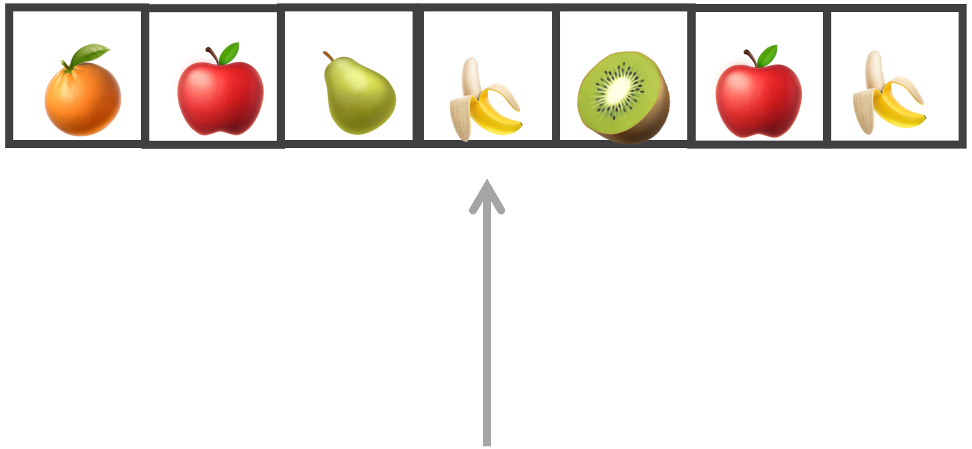
Count()

```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']


>>> fruits.count('apple')
2

>>> fruits.count('tangerine')
1
```



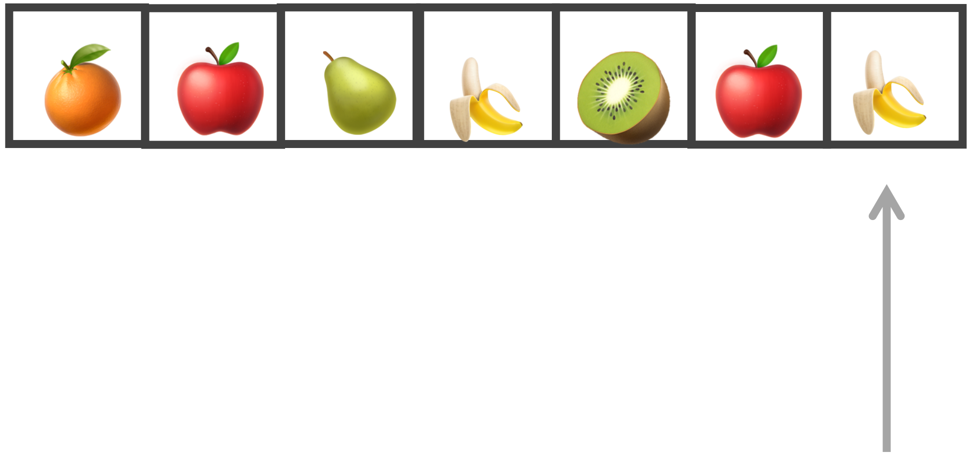
Index()

```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']


>>> fruits.index('banana')
3
```



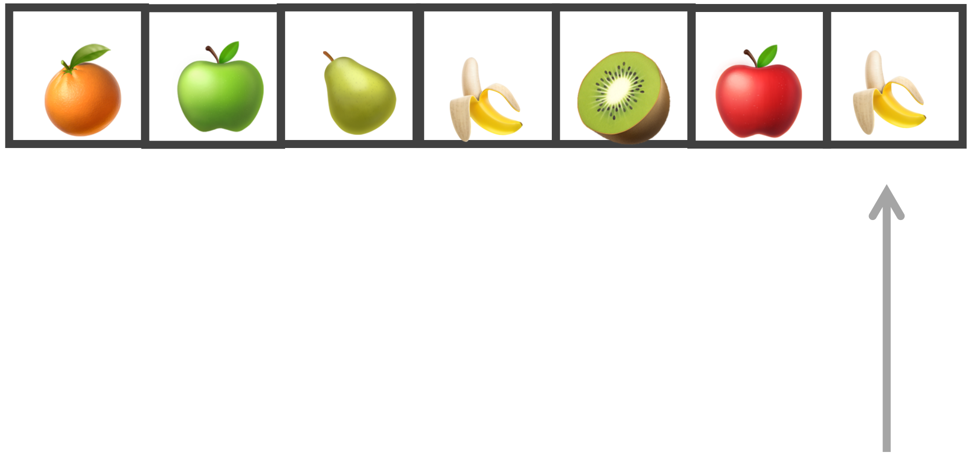
Index()

```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']


>>> fruits.index('banana')
3

>>> fruits.index('banana', 4) # Find next
banana starting a position 4
6
```



Index()

```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']

>>> fruits.index('banana')
3

>>> fruits.index('banana', 4) # Find next
banana starting a position 4
6
```

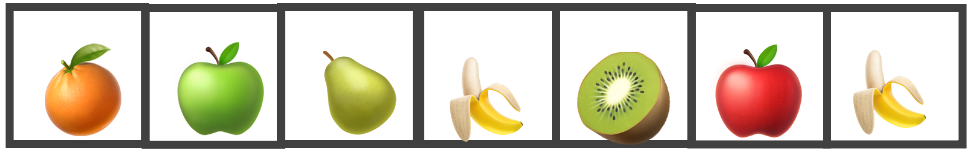


Reverse()

```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']

>>> fruits.reverse()
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'tangerine']
```

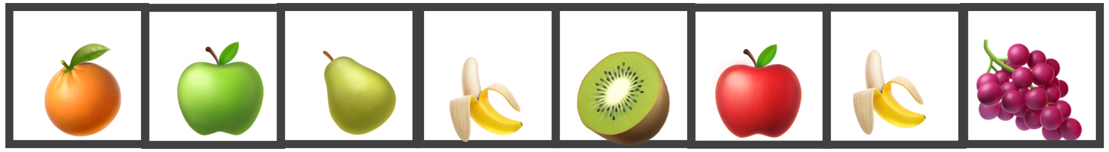
List methods

```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```



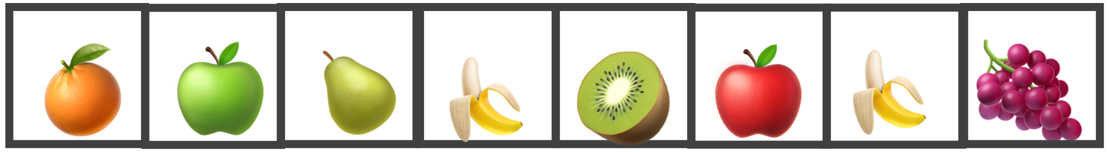
List methods – append()

```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']

>>> fruits.append('grape')
>>> fruits
['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana', 'grape']
```



List methods – pop()

```
>>> fruits.pop('grape')
```

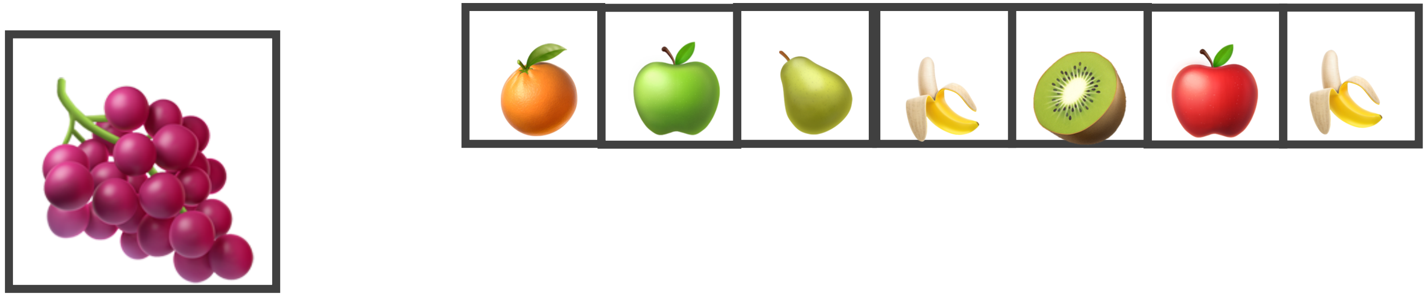


List methods – pop()

```
>>> fruits.pop('grape')
>>> fruits
['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

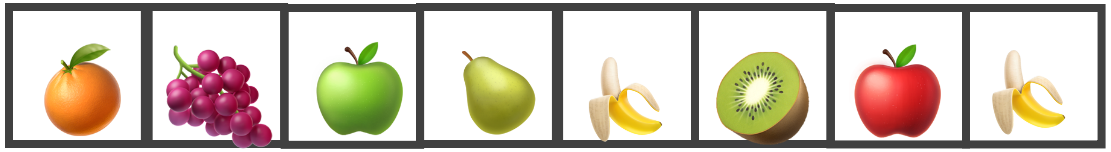
List methods

```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```



List methods – insert()

```
# Make a list of fruits
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']

>>> fruits.insert(1,'grape')
>>> fruits
['tangerine', 'grape', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

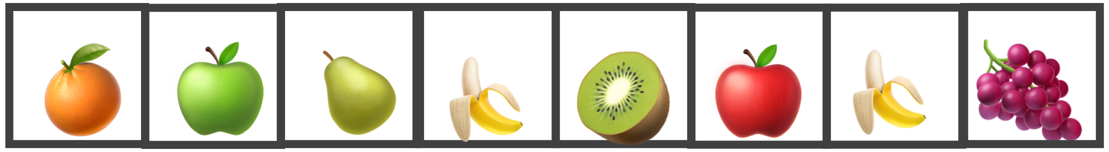


List methods – remove()

```
>>> fruits.remove('grape')
>>> fruits
['tangerine','apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

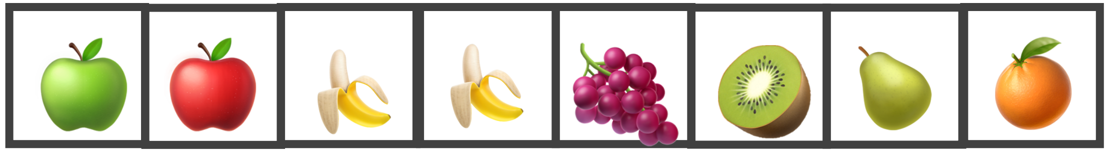
List methods – sort()

```
>>> fruits.sort()
```



List methods – sort()

```
>>> fruits.sort()
>>> fruits
['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'pear', 'tangerine']
```



List methods

```
>>> fruits = ['tangerine', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> fruits.count('apple')
2
>>> fruits.count('tangerine')
1
>>> fruits.index('banana')
3
>>> fruits.index('banana', 4) # Find next banana starting a position 4
6
>>> fruits.reverse()
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'tangerine']
>>> fruits.append('grape')
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'tangerine', 'grape']
>>> fruits.sort()
>>> fruits
['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'pear', 'tangerine']
>>> fruits.pop()
'tangerine'
```

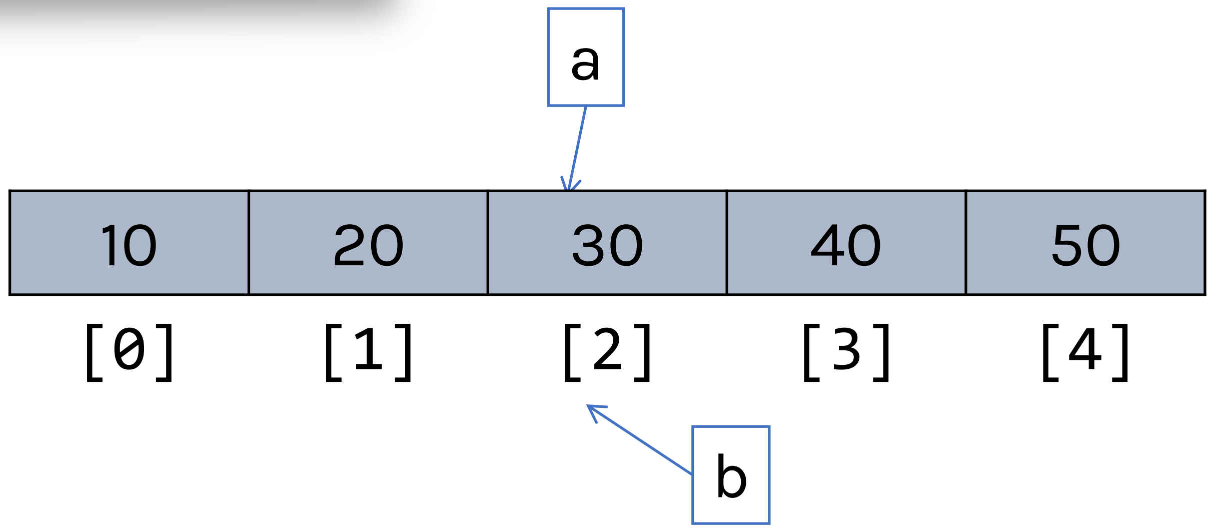
Copying a List

List Aliasing

- Aliasing means giving another name to the existing object. It

Modification in *a* will affect *b* and vice versa.

- b = a

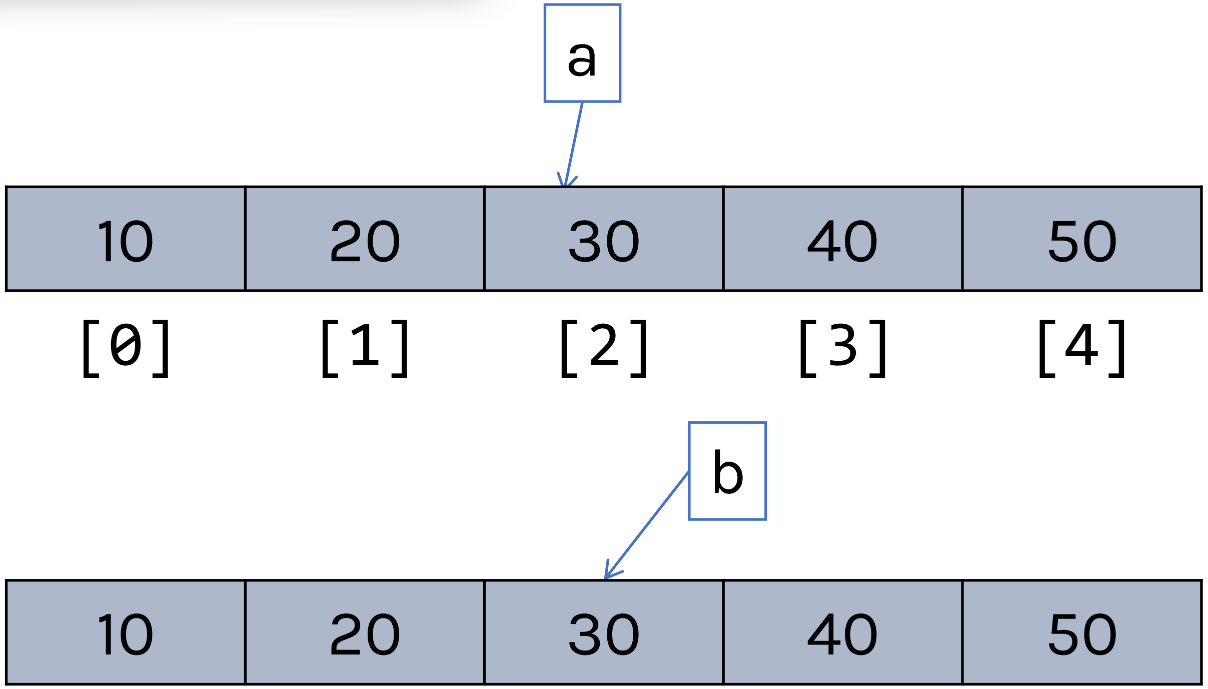


10	20	30	40	50
[0]	[1]	[2]	[3]	[4]

Copy a list

- W of all the elements is stored in ependent.
- a
- `b = a.copy()`

Modification in *a* will **not** affect *b* and vice versa.

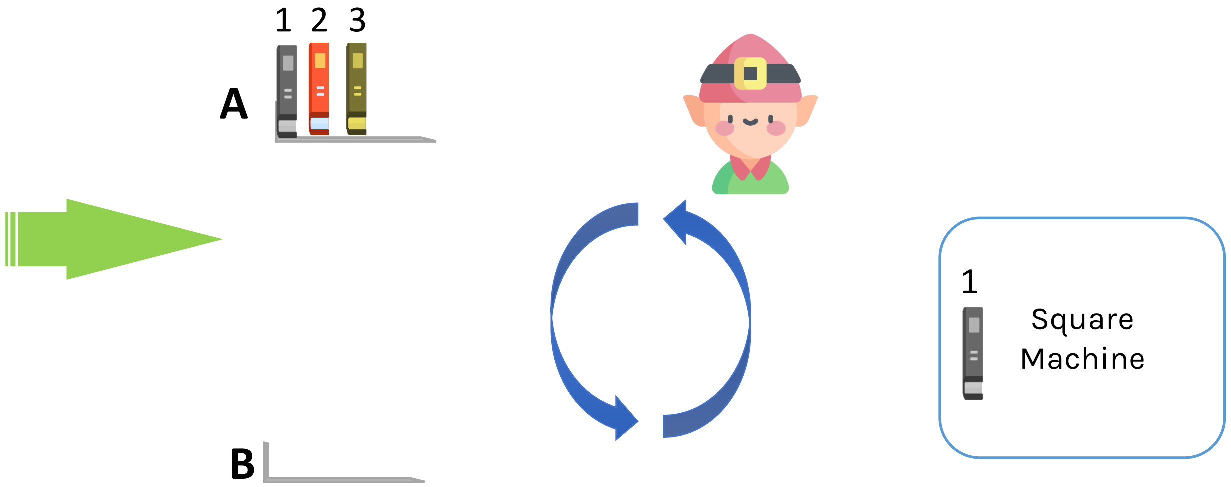


List Comprehension

List Comprehension is a Pythonic way for making lists and loops.

List = [expression for item in iterable]

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a For Loop to get a list containing their
# squares
>>> B = []

>>> for number in A:
....        B.append(number**2)

>>> print(B)
[2,4,9,16]
```

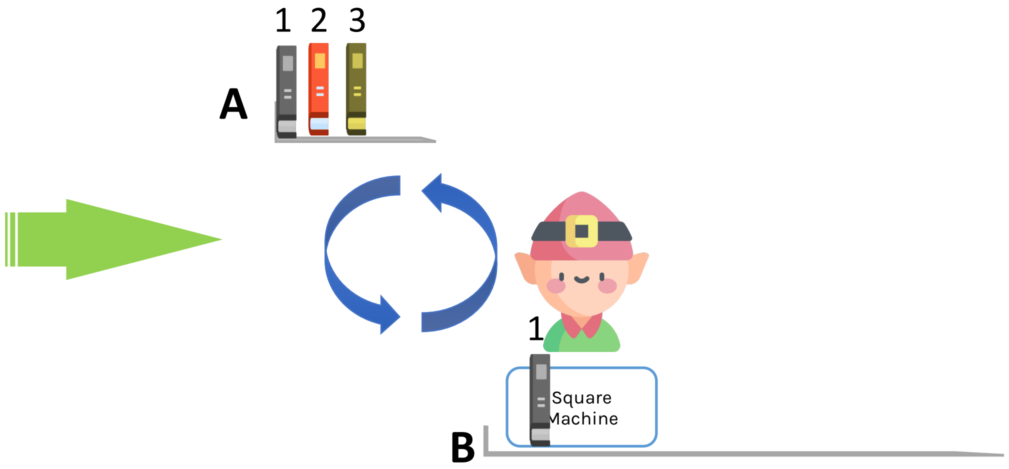


List Comprehension

List Comprehension is a Pythonic way for making lists and loops.

List = [expression for item in iterable]

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a List Comprehension to get a list
containing their # squares
>>> B = [ number**2 for number in A ]

>>> print(B)
[2,4,9,16]
```



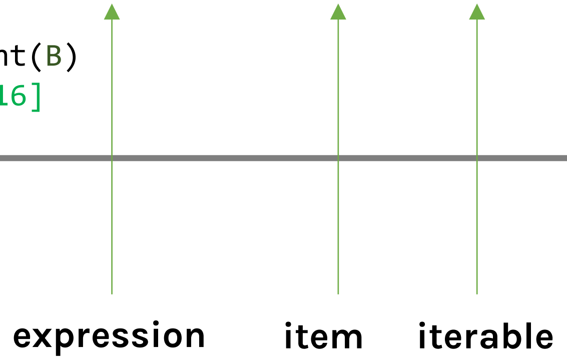
List Comprehension

List Comprehension is a Pythonic way for making lists and loops.

List = [expression **for** item **in** iterable]

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a For Loop to get a list containing their
# squares
>>> B = []

>>> for number in A:
....        B.append(number**2)

>>> print(B)
[2,4,9,16]
```

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a List Comprehension to get a list
containing their # squares
>>> B = [ number**2 for number in A ]

>>> print(B)
[2,4,9,16]
```



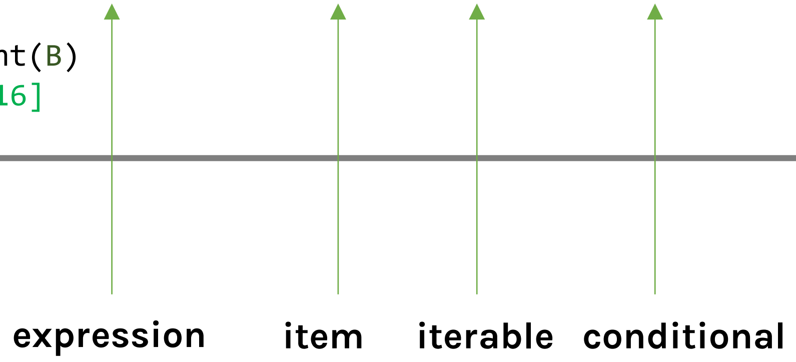
List Comprehension

List Comprehension with if conditional

List = [expression **for** item **in** iterable **if** conditional]

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a For Loop to get a list containing their
# squares – Condition: square the number only if
even
>>> B = []

>>> for number in A:
....         if number % 2 == 0:
....                  B.append(number**2)

>>> print(B)
[4,16]
```

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a List Comprehension to get a list containing
their squares – condition: even number
>>> B = [ number**2 for number in A if number %2 == 0]

>>> print(B)
[2,4,9,16]
```

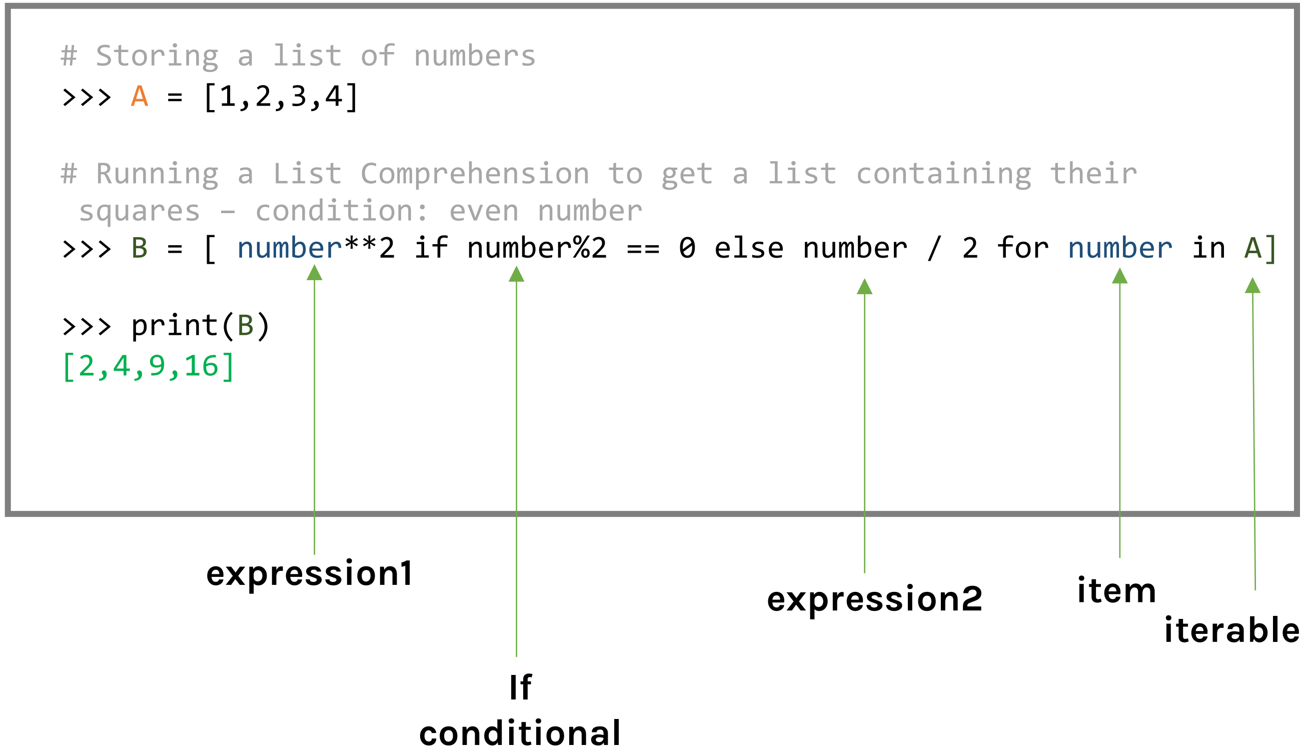


List Comprehension

List Comprehension with if & else conditional

List = [expression1 (**if conditional**) **else** expression2 **for** item **in** iterable]

```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a For Loop to get a list
#containing their # squares – Condition: square the
number only if even, else divide the number by 2
>>> B = []

>>> for number in A:
....         if number % 2 == 0:
....                  B.append(number**2)
....   else:
....       B.append(number / 2)

>>> print(B)
[4,16]
```



```
# Storing a list of numbers
>>> A = [1,2,3,4]

# Running a List Comprehension to get a list containing their
 squares – condition: even number
>>> B = [ number**2 if number%2 == 0 else number / 2 for number in A]

>>> print(B)
[2,4,9,16]
```