MODULE 6: LISTS

Instructor: Jonny C.H Yu

Sequence (序列) Data Structure

A data structure contains multiple items of data. **The items are stored in sequence one after another.**

Examples: your name, contact list, shopping list, music play list, your computer codes, poker cards, mathematic series, waiting queue, documents pile up on your desk etc.

SEQUENCES

- **Sequence: an object that contains multiple items of data**
  - The items are stored in sequence one after another
- **Python provides different types of sequences, including lists, tuples and strings**
  - The difference between lists and tuples is that a list is mutable and a tuple is immutable

INTRODUCTION TO LISTS

- **List: an object that contains multiple data items**
  - Element: An item in a list
  - Format: `list = [item1, item2, etc.]`
  - Can hold items of different types

Figure 7-1 A list of integers

Figure 7-2 A list of strings

Figure 7-3 A list holding different types

INTRODUCTION TO LISTS

- Many of the capabilities shown in this section apply to all sequence types.

Creating a List

- **Lists** typically store **homogeneous data**, but may store **heterogeneous data**.

```
[1]: c = [-45, 6, 0, 72, 1543]
```

```
[2]: c
```

```
[2]: [-45, 6, 0, 72, 1543]
```

M6_List_Intro.ipynb

In Codes_Module06.zip

Accessing Elements of a List

- Reference a list element by writing the list's name followed by the element's **index** enclosed in `[]` (the **subscription operator**).

Position number (2) of this element within the sequence

EXAMPLE: MANIPULATE A SIMPLE SERIES

- **Create a list that contains a sequence from 1 to N, with N specified by the user. Print the list and report the sum, average, max and min. Below is a sample run:**

```
Create a sequence from 1 to N; enter N: 10
The sequence is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum, average, max and min are 55 5.5 10 1
```

```
N = int(input('Create a sequence from 1 to N; enter N:'))
```

```
Create a sequence from 1 to N; enter N: 10
```

```
a_list = []
for i in range(1, N+1):
    a_list += [i]
```

```
print('The sequence is:', a_list)
```

```
The sequence is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
print('Sum, average, max and min are', sum(a_list), sum(a_list)/len(a_list), max(a_list), min(a_list))
```

```
Sum, average, max and min are 55 5.5 10 1
```

EXERCISE: SENTINEL AVERAGE

- **Prompt the user to enter a few integers with ‘q’ or ‘Q’ to quit. Report the average, max and min. Below is a sample run:**

```
Enter an integer (q or Q to quit): 5
Enter an integer (q or Q to quit): 8
Enter an integer (q or Q to quit): Q
Average, max and min are 6.5 8 5
```

EXERCISE: SENTINEL AVERAGE (ANS)

- **Prompt the user to enter a few integers with 'q' or 'Q' to quit. Report the average, max and min. Below is a sample run:**

```
Enter an integer (q or Q to quit): 5
Enter an integer (q or Q to quit): 8
Enter an integer (q or Q to quit): Q
Average, max and min are 6.5 8 5
```

```
a_list = []
```

```
s = input('Enter an integer (q or Q to quit):')
while (s != 'Q' and s != 'q'):
    a_list += [int(s)]
    s = input('Enter an integer (q or Q to quit):')
```

```
Enter an integer (q or Q to quit): 5
Enter an integer (q or Q to quit): 8
Enter an integer (q or Q to quit): Q
```

```
print('Average, max and min are', sum(a_list)/len(a_list), max(a_list), min(a_list))
```

```
Average, max and min are 6.5 8 5
```

FUN EXAMPLE: SHUFFLE YOUR SONGS

- Simulate randomly shuffling of yo

```
[1]: import random
```

```
[2]: songs = ['Faded', 'Fake a Smile', 'Alone, Pt. II',
              'On My Way', 'Lily']
```

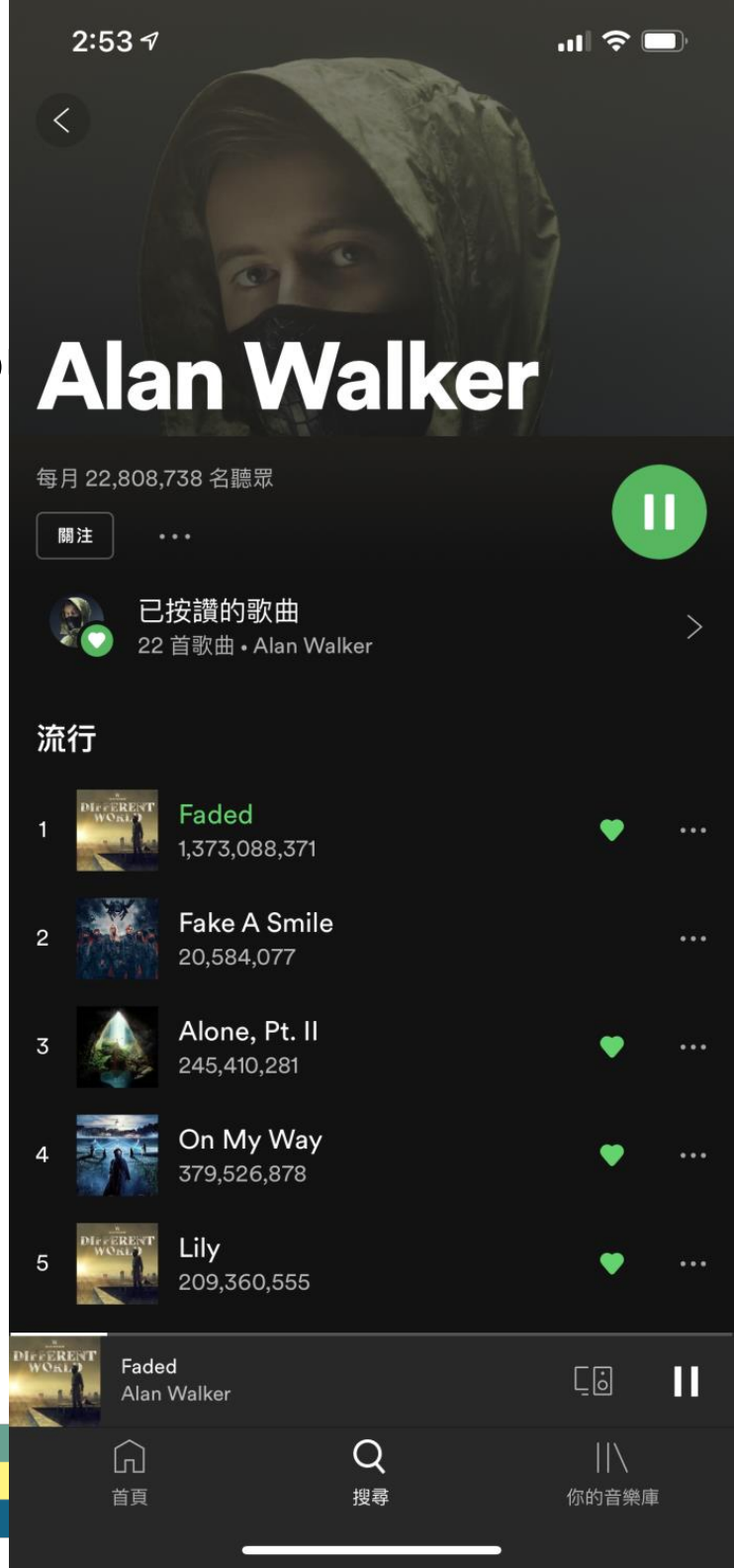
```
[3]: print(songs)
```

```
['Faded', 'Fake a Smile', 'Alone, Pt. II', 'On My Way', 'Lily']
```

```
[4]: random.shuffle(songs)
```

```
[5]: print(songs)
```

```
['Faded', 'Lily', 'Alone, Pt. II', 'On My Way', 'Fake a Smile']
```



LIST SLICING (: NOTATION)

- **Slice: a span of items that are taken from a sequence**
  - List slicing format: `list[start : end]`
  - Span is a list containing copies of elements from `start` up to, but not including, `end`
    - If `start` not specified, `0` is used for start index
    - If `end` not specified, `len(list)` is used for end index
  - Slicing expressions can include a step value and negative indexes relative to end of list

LIST SLICING

- Can **slice** sequences to create new sequences of the same type containing *subsets* of the original elements.
- Slice operations that do *not* modify a sequence work identically for lists, tuples and strings.

Specifying a Slice with Starting and Ending Indices

```
[1]: numbers = [2, 3, 5, 7, 11, 13, 17, 19]
```

```
[2]: numbers[2:6]
```

```
[2]: [5, 7, 11, 13]
```

M6_List_Slice.ipynb

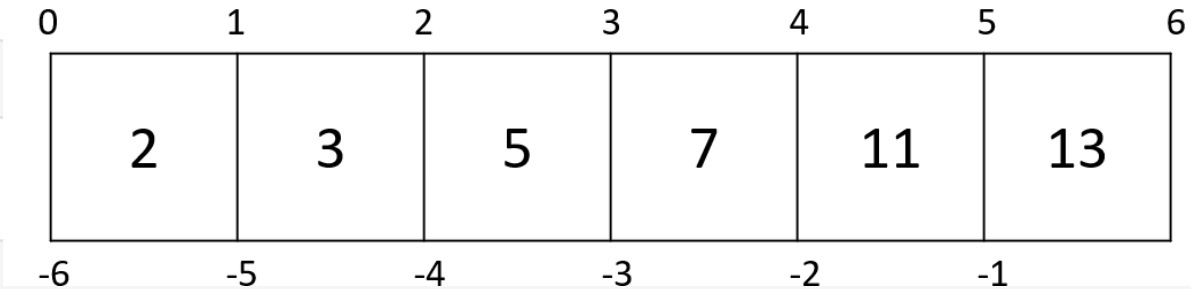
Specifying a Slice with Only an Ending Index

- Starting index `0` is assumed.

```
[3]: numbers[:6]
```

```
[3]: [2, 3, 5, 7, 11, 13]
```

```
[4]: numbers[0:6]
```



EXERCISE: LIST SLICING

Create a list containing the values from 1 to 15

[1]:
```
a_list = list(range(1, 16))
```

[2]:
```
a_list
```

[2]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

Use slicing to select a_list even number

[3]:
```

```

[3]:
```
[2, 4, 6, 8, 10, 12, 14]
```

Replace the elements at indices 5 through 9 with 0, then show the resulting list

[4]:
```

```

[5]:
```
a_list
```

[5]:
```
[1, 2, 3, 4, 5, 0, 0, 0, 0, 11, 12, 13, 14, 15]
```

Keep only the first five elements, then show the resulting list

[6]:
```

```

[7]:
```
a_list
```

[7]:
```
[1, 2, 3, 4, 5]
```

EXERCISE: LIST SLICING (ANS)

Create a list containing the values from 1 to 15

```
[1]: a_list = list(range(1, 16))
```

```
[2]: a_list
```

```
[2]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

Use slicing to select a_list even number

```
[3]: a_list[1::2]
```

```
[3]: [2, 4, 6, 8, 10, 12, 14]
```

Replace the elements at indices 5 through 9 with 0, then show the resulting list

```
[4]: a_list[5:10] = [0, 0, 0, 0]
```

```
[5]: a_list
```

```
[5]: [1, 2, 3, 4, 5, 0, 0, 0, 0, 11, 12, 13, 14, 15]
```

Keep only the first five elements, then show the resulting list

```
[6]: a_list[5:] = []
```

```
[7]: a_list
```

```
[7]: [1, 2, 3, 4, 5]
```

SOME LIST METHODS

Table 7-1 A few of the list methods

Method	Description
`append(item)`	Adds *item* to the end of the list.
`index(item)`	Returns the index of the first element whose value is equal to item. A `ValueError` exception is raised if item is not found in the list.
`insert(index, item)`	Inserts *item* into the list at the specified *index*. When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at the specified index, and all the items after it, are shifted by one position toward the end of the list. No exceptions will occur if you specify an invalid index. If you specify an index beyond the end of the list, the item will be added to the end of the list. If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list.
`sort()`	Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value).
`remove(item)`	Removes the first occurrence of *item* from the list. A `ValueError` exception is raised if item is not found in the list.
`reverse()`	Reverses the order of the items in the list.

SOME LIST METHODS

```
[1]: color_names = ['orange', 'yellow', 'green']
```

Inserting an Element at a Specific List Index

```
[2]: color_names.insert(0, 'red')
```

```
[3]: color_names
```

```
[3]: ['red', 'orange', 'yellow', 'green']
```

M6_Some_List_Methods.ipynb

Adding an Element to the End of a List

```
[4]: color_names.append('blue')
```

```
[5]: color_names
```

```
[5]: ['red', 'orange', 'yellow', 'green', 'blue']
```

Adding All the Elements of a Sequence to the End of a List

EXERCISE: SENTINEL AVERAGE TAKE 2

- **Prompt the user to enter a few integers with ‘q’ or ‘Q’ to quit. Now use append() method to add the item into your list. Report the average, max and min. Below is a sample run:**

```
Enter an integer (q or Q to quit): 5
Enter an integer (q or Q to quit): 8
Enter an integer (q or Q to quit): Q
Average, max and min are 6.5 8 5
```

EXERCISE: SENTINEL AVERAGE TAKE 2 (ANS)

- **Prompt the user to enter a few integers with ‘q’ or ‘Q’ to quit. Use append() method to add the item into your list. Report the average, max and min.**

```
[1]: a_list = []

[2]: s = input('Enter an integer (q or Q to quit):')
     while (s != 'Q' and s != 'q'):
         a_list.append(int(s))
         s = input('Enter an integer (q or Q to quit):')

     Enter an integer (q or Q to quit): 5
     Enter an integer (q or Q to quit): 4
     Enter an integer (q or Q to quit): 3
     Enter an integer (q or Q to quit): q

[3]: print('Average, max and min are', sum(a_list)/len(a_list), max(a_list), min(a_list))

     Average, max and min are 4.0 5 3
```

EXERCISE: REVERSE A NUMBER

- List makes complex logic easy. Prompt the user to enter an integer and display it in a reverse order

```
[1]: def reverse(number):
```

```
[2]: number = int(input('Enter a number:'))
```

```
Enter a number: 543
```

```
[3]: reverse(number)
```

```
345
```

EXERCISE: REVERSE A NUMBER (ANS)

- **List makes complex logic easy. Prompt the user to enter an integer and display it in a reverse order**

[1]:
```
def reverse(number):
    a_list = []
    a_list.extend(str(number))
    for i in a_list[::-1]:
        print(i, end = "")
```

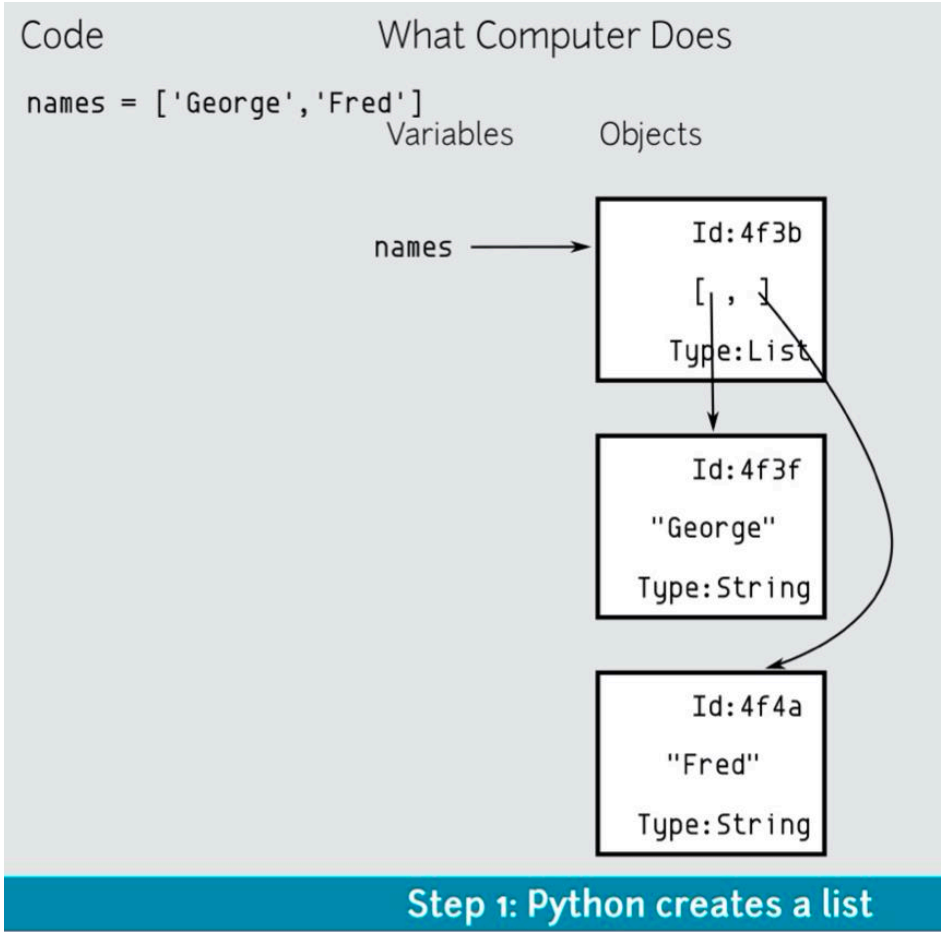
[2]:
```
number = int(input('Enter a number:'))
```

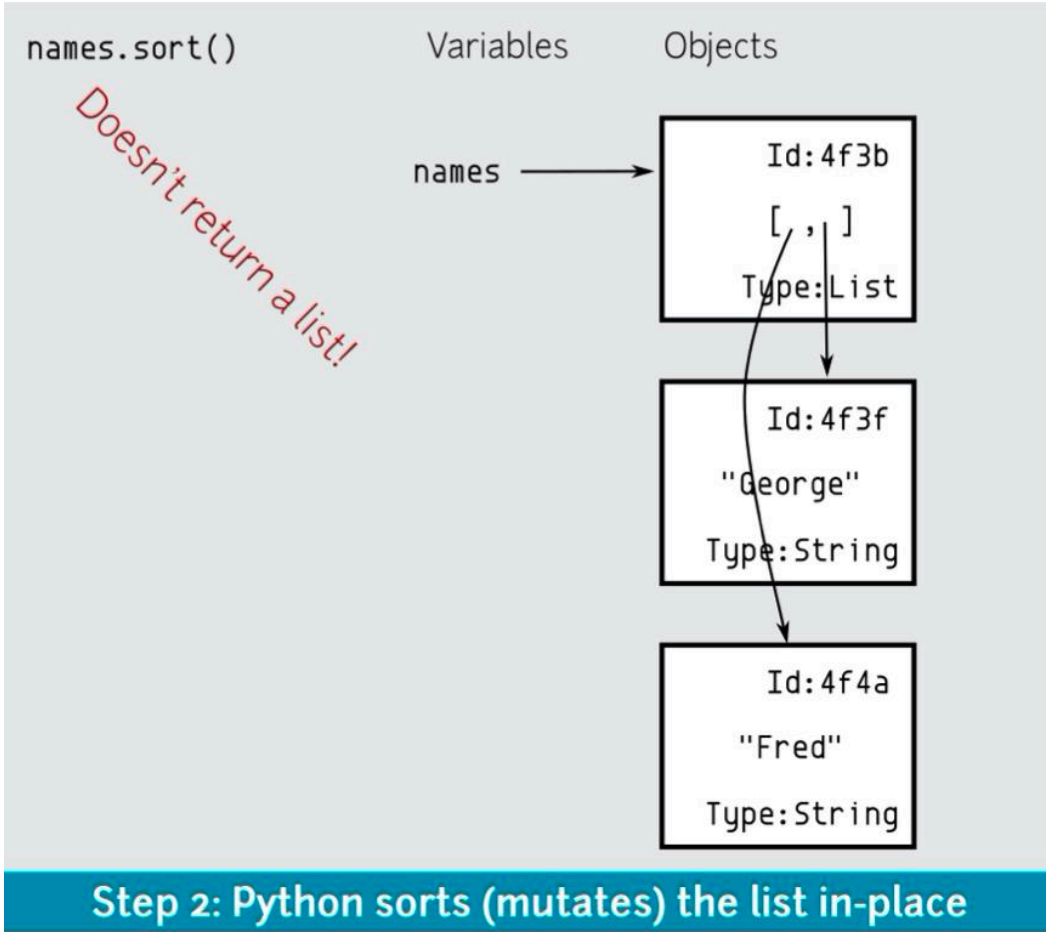
```
Enter a number: 543
```

[3]:
```
reverse(number)
```

```
345
```

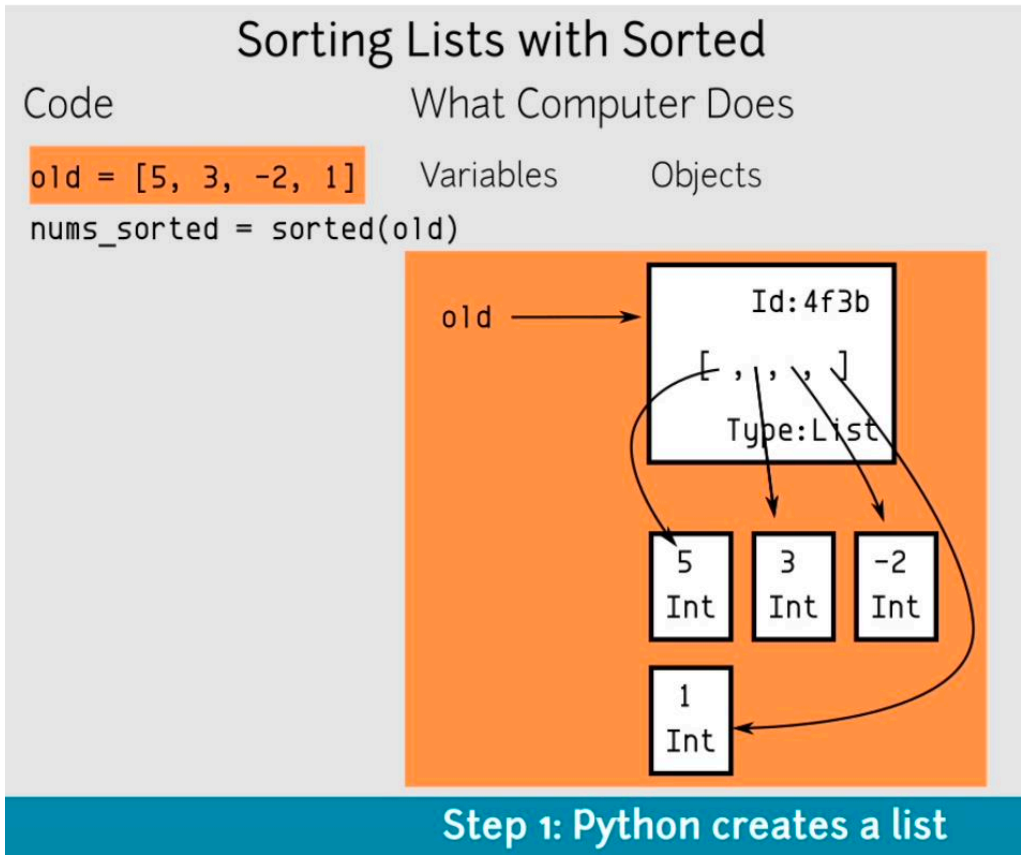
LIST SORT METHOD

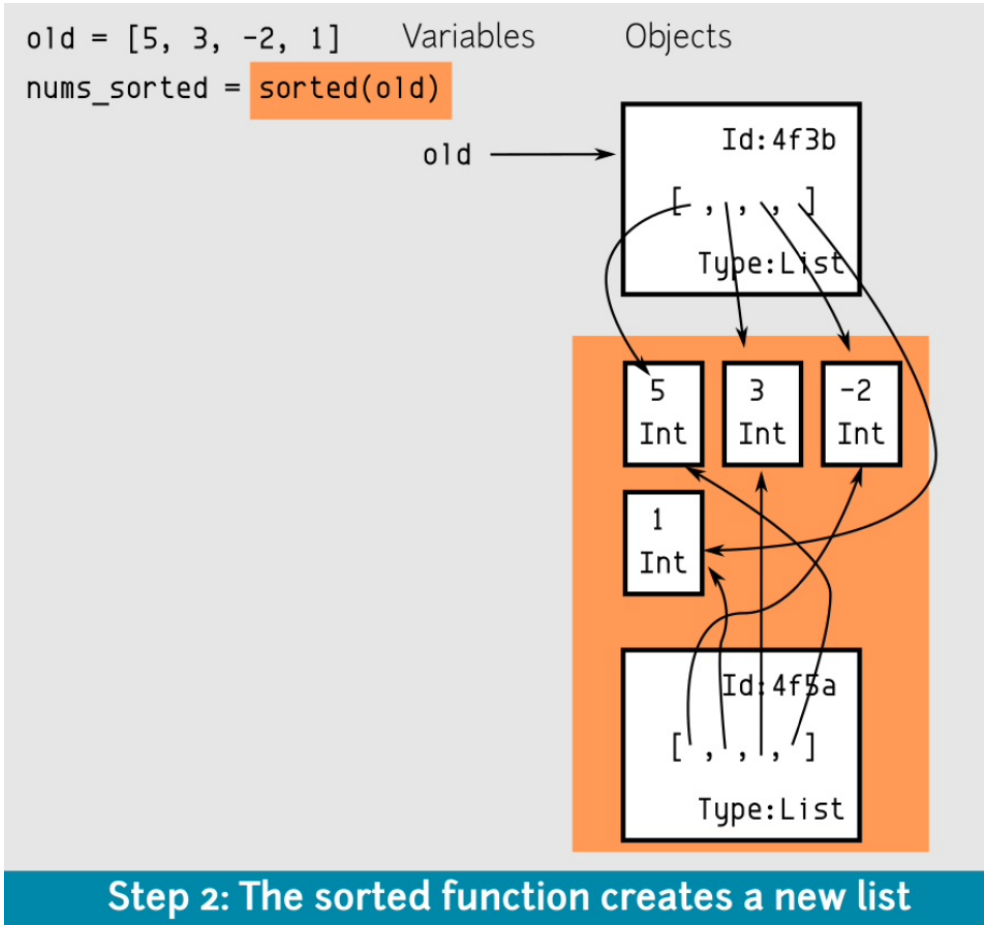




Will change the original list.
Return None.

BUILT-IN sorted





Will not change the original list.
Return a sorted list.

SORTING LIST

Sorting a List in Ascending Order

- List method `sort` *modifies* a list.

```
[1]: numbers = [10, 3, 7, 1, 9, 4, 2, 8, 5, 6]

[2]: numbers.sort()

[3]: numbers

[3]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

M6_Sorting.ipynb

Sorting a List in Descending Order

```
[4]: numbers.sort(reverse=True)

[5]: numbers

[5]: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Built-In Function `sorted`

EXERCISE: SHUFFLING AND SORTING SERIES

- **Create a list that contains a sequence from 1 to N, with N specified by the user. Print the list, shuffle the list and sort the list in ascending and descending orders. Below is a sample run:**

```
Create a sequence from 1 to N; enter N: 10
The original sequence is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
The sequence after shuffling is: [4, 8, 3, 7, 2, 1, 6, 9,
5, 10]
The sequence in an ascending order is: [1, 2, 3, 4, 5, 6,
7, 8, 9, 10]
The sequence in a descending order is: [10, 9, 8, 7, 6,
5, 4, 3, 2, 1]
```

EXERCISE: SHUFFLING AND SORTING SERIES (ANS)

[1]:
```
N = int(input('Create a sequence from 1 to N; enter N:'))
```

```
Create a sequence from 1 to N; enter N: 10
```

[2]:
```
a_list = []
for i in range(1, N+1):
    a_list += [i]
```

[3]:
```
print('The original sequence is:', a_list)
```

```
The original sequence is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[4]:
```
import random
random.shuffle(a_list)
print('The sequence after shuffling is:', a_list)
```

```
The sequence after shuffling is: [4, 8, 3, 7, 2, 1, 6, 9, 5, 10]
```

[5]:
```
a_list.sort()
print('The sequence in an ascending order is:', a_list)
```

```
The sequence in an ascending order is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[6]:
```
a_list.sort(reverse = True)
print('The sequence in an descending order is:', a_list)
```

```
The sequence in an descending order is: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

SEARCHING LIST

- **Searching** is the process of locating a particular **key** value.

List Method `index`

- Searches through a list from index 0 and returns the index of the *first* element that matches the search key.
- `ValueError` if the value is not in the list.

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

```
[2]: numbers.index(5)
```

M6_Searching.ipynb

```
[2]: 6
```

Specifying the Starting Index of a Search

```
[3]: numbers *= 2
```

```
[4]: numbers
```

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

EXERCISE: SAFE SEARCH WITH INDEX METHOD AND IN OPERATOR

- **Create** a five-element list containing 67, 12, 46, 43 and 13, then use list method index to search for a 43 and 44. Ensure that no ValueError occurs when searching for 44.

```
[1]: def index_found(a_list, element):

[2]: numbers = [67, 12, 46, 43, 13]

[3]: index_found(numbers, 43)
     Found 43 at index: 3

[4]: index_found(numbers, 44)
     44 not found
```

EXERCISE: SAFE SEARCH WITH INDEX METHOD AND IN OPERATOR (ANS)

- **Create** a five-element list containing 67, 12, 46, 43 and 13, then use list method index to search for a 43 and 44. Ensure that no ValueError occurs when searching for 44.

[1]:
```
def index_found(a_list, element):
    if element in a_list:
        print('Found', element, 'at index:', a_list.index(element))
    else:
        print(element, 'not found')
```

[2]:
```
numbers = [67, 12, 46, 43, 13]
```

[3]:
```
index_found(numbers, 43)
```

```
Found 43 at index: 3
```

[4]:
```
index_found(numbers, 44)
```

```
44 not found
```

`del` STATEMENT

- **`del` statement: removes an element from a specific index in a list**
  - General format: `del list[i]`
  - Often use together with slicing

Deleting the Element at a Specific List Index

```
[1]: numbers = list(range(0, 10))

[2]: numbers

[2]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[3]: del numbers[-1]

[4]: numbers

[4]: [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

M6_del.ipynb

Deleting a Slice from a List

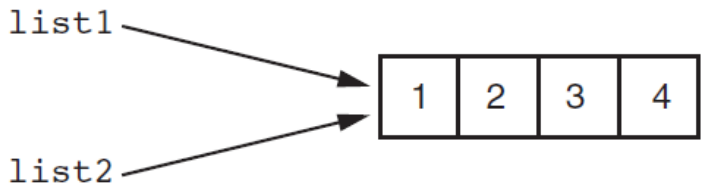
See Note_M6.pdf

```
[5]: del numbers[0:2]
```

COPYING LISTS

- **To make a copy of a list you must copy each element of the list**

Figure 7-4 `list1` and `list2` reference the same list




```
[1]: list1 = [1, 2, 3, 4]
     list2 = list1
     print(id(list1), id(list2))
```

```
4398870848 4398870848
```

```
[2]: list1[0] = 99
     print ('list1:', list1)
     print ('list2:', list2)
```

```
list1: [99, 2, 3, 4]
list2: [99, 2, 3, 4]
```

COPYING LISTS (CONT'D.)

- **To make a copy of a list you must copy each element of the list**
  - Two methods to do this:
    - Creating a new empty list and using a `for` loop to add a copy of each element from the original list to the new list
    - Creating a new empty list and concatenating the old list to the new empty list

```
[1]: list1 = [1, 2, 3, 4]
     list2 = [] + list1
     print(id(list1), id(list2))

     4420969648 4420959760

[2]: list1[0] = 99
     print ('list1:', list1)
     print ('list2:', list2)

     list1: [99, 2, 3, 4]
     list2: [1, 2, 3, 4]
```

List Operations

You can see lists are very powerful data structures, with possibilities that go far beyond those of plain old arrays. List operations are very important in Python programming.

Being familiar with these list operations will make your life as a Python coder much easier.

List operation	Explanation	Example
`[]`	Creates an empty list	`x = []`
`len`	Returns the length of a list	`len(x)`
`append`	Adds a single element to the end of a list	`x.append('y')`
`extend`	Adds another list to the end of the list	`x.extend(['a', 'b'])`
`insert`	Inserts a new element at a given position in the list	`x.insert(0, 'y')`
`del`	Removes a list element or slice	`del(x[0])`
`remove`	Searches for and removes a given value from a list	`x.remove('y')`
`reverse`	Reverses a list in place	`x.reverse()`
`sort`	Sorts a list in place	`x.sort()`
`+`	Adds two lists together	`x1 + x2`
`*`	Replicates a list	`x = ['y'] * 3`
`min`	Returns the smallest element in a list	`min(x)`
`max`	Returns the largest element in a list	`max(x)`
`index`	Returns the position of a value in a list	`x.index['y']`
`count`	Counts the number of times a value occurs in a list	`x.count('y')`

List operation	Explanation	Example
`sum`	Sums the items (if they can be summed)	`sum(x)`
`in`	Returns whether an item is in a list	`'y' in x`

SUMMARY

- This module covered:
  - The basic concept of lists
  - Basic techniques for creating and processing lists
  - Slicing and copying lists
  - Useful built-in functions for lists
  - Some list methods
  - Sorting and searching lists