In [1]:
```
help(list.sort)
```

```
Help on method_descriptor:

sort(self, /, *, key=None, reverse=False)
    Sort the list in ascending order and return None.

    The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
    order of two equal elements is maintained).

    If a key function is given, apply it once to each list item and sort them,
    ascending or descending, according to their function values.

    The reverse flag can be set to sort in descending order.
```

In [2]:
```
# sort() - sorts the list ascending by default

courses = ["bca", "MCa", "btech", "bsc", "msc"]
courses.sort()
print(f"courses sorted in ascending order: {courses}")

scores = [23, 29, 19, 9]
scores.sort()
print(f"scores sorted in ascending order: {scores}")
```

```
courses sorted in ascending order: ['MCa', 'bca', 'bsc', 'btech', 'msc']
scores sorted in ascending order: [9, 19, 23, 29]
```

In [3]:
```
# sort() - sorts in descending order by setting the 'reverse' flag

courses = ["bca", "MCa", "btech", "bsc", "msc"]
courses.sort(reverse = True)
print(f"courses sorted in descending order: {courses}")
```

```
courses sorted in descending order: ['msc', 'btech', 'bsc', 'bca', 'MCa']
```

In [4]:
```
help(sorted)
```

```
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

In [5]:
```
# original list kept intact. Return a new list with sorted()

courses = ["bca", "MCa", "btech", "bsc", "msc"]
x = sorted(courses)
print(f"courses: {courses}")
print(f"x : {x}")
```

```
courses: ['bca', 'MCa', 'btech', 'bsc', 'msc']
x : ['MCa', 'bca', 'bsc', 'btech', 'msc']
```

In [6]:
```
# original list kept intact. copy created by slicing and sorting applied on it

courses = ["bca", "MCa", "btech", "bsc", "msc"]
x = courses[:]
x.sort()
print(f"courses: {courses}")
print(f"x : {x}")
```

```
courses: ['bca', 'MCa', 'btech', 'bsc', 'msc']
x : ['MCa', 'bca', 'bsc', 'btech', 'msc']
```

In [7]:
```
help(id)
```

```
Help on built-in function id in module builtins:

id(obj, /)
    Return the identity of an object.

    This is guaranteed to be unique among simultaneously existing objects.
    (CPython uses the object's memory address.)
```

In [11]:
```
# assignment: creating alias
# Assignment statements in Python do not copy objects,
# they create bindings between a target and an object.

x = [11, 22, [33, 44, 66], 55]
y = x

print(f"id(x)={id(x)}, x: {x}")
print(f"id(y)={id(y)}, y: {y}")

y.append(3)
print(f"id(x)={id(x)}, x: {x}")
print(f"id(y)={id(y)}, y: {y}")
```

```
y[1] = 99
y[2][0] = 100
print(f"id(x)={id(x)}, x: {x}")
print(f"id(y)={id(y)}, y: {y}")
```

```
id(x)=2249317268352, x: [11, 22, [33, 44, 66], 55]
id(y)=2249317268352, y: [11, 22, [33, 44, 66], 55]
id(x)=2249317268352, x: [11, 22, [33, 44, 66], 55, 3]
id(y)=2249317268352, y: [11, 22, [33, 44, 66], 55, 3]
id(x)=2249317268352, x: [11, 99, [100, 44, 66], 55, 3]
id(y)=2249317268352, y: [11, 99, [100, 44, 66], 55, 3]
```

In [12]:
```
# For collections that are mutable or contain mutable items,
# a copy is sometimes needed so one can change one copy without changing the other.

# Shallow copy vs. deep copy

# The difference between shallow and deep copying is only relevant for
# compound objects (objects that contain other objects, like lists or class instances)

# A shallow copy constructs a new compound object and then (to the extent possible)
# inserts references into it to the objects found in the original.

# A deep copy constructs a new compound object and then, recursively,
# inserts copies into it of the objects found in the original.
```

In [13]:
```
# shallow copy
# slicing

x = [11, 22, [33, 44, 66], 55]
y = x[:]

y[1] = 99
y[2][0] = 100

print(f"y: {y}")
print(f"x: {x}")
```

```
y: [11, 99, [100, 44, 66], 55]
x: [11, 22, [100, 44, 66], 55]
```

In [14]:
```
# shallow copy
# list.copy()

x = [11, 22, [33, 44, 66], 55]
y = x.copy()

y[1] = 99
y[2][0] = 100

print(f"y: {y}")
print(f"x: {x}")
```

```
y: [11, 99, [100, 44, 66], 55]
x: [11, 22, [100, 44, 66], 55]
```

In [15]:
```
# shallow copy
# list()

x = [11, 22, [33, 44, 66], 55]
y = list(x)

y[1] = 99
y[2][0] = 100

print(f"y: {y}")
print(f"x: {x}")
```

```
y: [11, 99, [100, 44, 66], 55]
x: [11, 22, [100, 44, 66], 55]
```

In [16]:
```
# shallow copy

import copy

x = [11, 22, [33, 44, 66], 55]
y = copy.copy(x)

y[1] = 99
y[2][0] = 100

print(f"y: {y}")
print(f"x: {x}")
```

```
y: [11, 99, [100, 44, 66], 55]
x: [11, 22, [100, 44, 66], 55]
```

In [17]:
```
# deep copy

import copy

x = [11, 22, [33, 44, 66], 55]
```

```
y = copy.deepcopy(x)

y[1] = 99
y[2][0] = 100

print(f"y: {y}")
print(f"x: {x}")
```

```
y: [11, 99, [100, 44, 66], 55]
x: [11, 22, [33, 44, 66], 55]
```

In [18]:
```
a = [15, 20]
b = [15, 20]
c = [10, a, b]
b[1] = 2*a[0]
c[1][0] = c[0]
c[0] = a[0] + c[1][1] + b[0] + c[2][1]
c[0]
```

Out[18]:
```
75
```

In [19]:
```
a = [15, 20]
b = a
c = [10, a, b]
b[1] = 2*a[0]
c[1][0] = c[0]
c[0] = a[0] + c[1][1] + b[0] + c[2][1]
c[0]
```

Out[19]:
```
80
```