shallow_and_deep_copy

November 11, 2023

[49]:
```
import copy
```

[50]:
```
xs = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
ys = list(xs)  # make a shallow copy
# ys contains references to the child objects stored in xs.
# when you modify one of the child objects in xs, this modification will be␣
 ↪reflected in ys as well-that's
#         because both lists share the same child objects. The copy is only a␣
 ↪"shallow"/"one level deep" copy.
```

[51]:
```
print(xs)
print(ys)
```

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

[52]:
```
# add a new sublist to the original (xs) and this didn't affect the copy (ys).
xs.append(['xy_0','xy_1'])
print(xs)
print(ys)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9], ['xy_0', 'xy_1']]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

[53]:
```
xs[1][0] = 'X'
xs[3][1] = 'xy_2'
print(xs)
print(ys)
```

```
[[1, 2, 3], ['X', 5, 6], [7, 8, 9], ['xy_0', 'xy_2']]
[[1, 2, 3], ['X', 5, 6], [7, 8, 9]]
```

[54]:
```
ys.append(['ys_0','ys_3'])
ys[1][0] = 'Y'
ys[2][1] = 'Y_1'
print(xs)
print(ys)
```

```
[[1, 2, 3], ['Y', 5, 6], [7, 'Y_1', 9], ['xy_0', 'xy_2']]
[[1, 2, 3], ['Y', 5, 6], [7, 'Y_1', 9], ['ys_0', 'ys_3']]
```

[ ]:

[16]:
```
######################################################################
xs = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
ys = xs.copy()  # make a shallow copy - another method.
```

[17]:
```
print(xs)
print(ys)
```

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

[18]:
```
# add a new sublist to the original (xs) and this didn't affect the copy (ys).
xs.append(['xy_0','xy_1'])
print(xs)
print(ys)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9], ['xy_0', 'xy_1']]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

[19]:
```
xs[1][0] = 'X'
xs[3][1] = 'xy_2'
print(xs)
print(ys)
```

```
[[1, 2, 3], ['X', 5, 6], [7, 8, 9], ['xy_0', 'xy_2']]
[[1, 2, 3], ['X', 5, 6], [7, 8, 9]]
```

[20]:
```
ys.append(['ys_0','ys_3'])
ys[1][0] = 'Y'
ys[2][1] = 'Y_1'
print(xs)
print(ys)
```

```
[[1, 2, 3], ['Y', 5, 6], [7, 'Y_1', 9], ['xy_0', 'xy_2']]
[[1, 2, 3], ['Y', 5, 6], [7, 'Y_1', 9], ['ys_0', 'ys_3']]
```

[21]:
```
######################################################################
xs = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
ys = copy.copy(xs)  # make a shallow copy - another method.
```

[22]:
```
print(xs)
print(ys)
```

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

[23]:
```
# add a new sublist to the original (xs) and this didn't affect the copy (ys).
xs.append(['xy_0','xy_1'])
print(xs)
print(ys)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9], ['xy_0', 'xy_1']]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

[24]:
```
xs[1][0] = 'X'
xs[3][1] = 'xy_2'
print(xs)
print(ys)
```

```
[[1, 2, 3], ['X', 5, 6], [7, 8, 9], ['xy_0', 'xy_2']]
[[1, 2, 3], ['X', 5, 6], [7, 8, 9]]
```

[25]:
```
ys.append(['ys_0','ys_3'])
ys[1][0] = 'Y'
ys[2][1] = 'Y_1'
print(xs)
print(ys)
```

```
[[1, 2, 3], ['Y', 5, 6], [7, 'Y_1', 9], ['xy_0', 'xy_2']]
[[1, 2, 3], ['Y', 5, 6], [7, 'Y_1', 9], ['ys_0', 'ys_3']]
```

[26]:
```
##########################################################################
xs = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
ys = copy.deepcopy(xs) # xs was cloned recursively, including all of its child␣
 ↪objects.
```

[27]:
```
print(xs)
print(ys)
```

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

[28]:
```
xs.append(['xy_0','xy_1'])
print(xs)
print(ys)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9], ['xy_0', 'xy_1']]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

[29]:
```
xs[1][0] = 'X'
xs[3][1] = 'xy_2'
```

```
print(xs)
print(ys)
```

```
[[1, 2, 3], ['X', 5, 6], [7, 8, 9], ['xy_0', 'xy_2']]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

[30]:
```
ys.append(['ys_0','ys_3'])
ys[1][0] = 'Y'
ys[2][1] = 'Y_1'
print(xs)
print(ys)
```

```
[[1, 2, 3], ['X', 5, 6], [7, 8, 9], ['xy_0', 'xy_2']]
[[1, 2, 3], ['Y', 5, 6], [7, 'Y_1', 9], ['ys_0', 'ys_3']]
```

[ ]:
```

```

[31]:
```
######################################################################3
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        '''string representation. !r is used inside an f-string as a formatting␣
 ↪specifier.
        !r is used to apply the repr formatting option to the expressions self.
 ↪x and self.y.

        converting the value to a string before calling format(), the normal␣
 ↪formatting logic is bypassed.
        '!s' which calls str() on the value, '!r' which calls repr() and '!a'␣
 ↪which calls ascii().

        repr() provides the official string representation of an object, aimed␣
 ↪at the programmer.
        str() provides the informal string representation of an object, aimed␣
 ↪at the user.'''
        return f'Point({self.x!r}, {self.y!r})'
```

[32]:
```
a = Point(23, 42)
b = copy.copy(a) # make a shallow copy.
```

[33]:
```
print(f"{a},", f"{b},", a is b)
# because our point object uses immutable types (ints) for its coordinates,␣
 ↪there's no difference between a
```

```
#                                                                    shallow␣
 ↪and a deep copy in this case.
```

```
Point(23, 42), Point(23, 42), False
```

[34]:
```
class Rectangle:
    def __init__(self, topleft, bottomright):
        self.topleft = topleft
        self.bottomright = bottomright

    def __repr__(self):
        return (f'Rectangle({self.topleft!r}, 'f'{self.bottomright!r})')
```

[35]:
```
rect = Rectangle(Point(0, 1), Point(5, 6))
srect = copy.copy(rect)
```

[36]:
```
print(f"{rect},\n", f"{srect},\n", rect is srect)
```

```
Rectangle(Point(0, 1), Point(5, 6)),
 Rectangle(Point(0, 1), Point(5, 6)),
 False
```

[37]:
```
rect.topleft.x = 999
print(f"{rect},\n", f"{srect}")
```

```
Rectangle(Point(999, 1), Point(5, 6)),
 Rectangle(Point(999, 1), Point(5, 6))
```

[38]:
```
drect = copy.deepcopy(srect)
drect.topleft.x = 222
print(f"{drect},\n",f"{rect},\n", f"{srect}")
```

```
Rectangle(Point(222, 1), Point(5, 6)),
 Rectangle(Point(999, 1), Point(5, 6)),
 Rectangle(Point(999, 1), Point(5, 6))
```

[39]:
```
################################################################################
a = [1, 2, 3]
b = [4, 5, 6]
c = [a, b]
c
```

[39]:
```
[[1, 2, 3], [4, 5, 6]]
```

[40]:
```
# id() => Return the "identity" of an object. This is an integer which is␣
 ↪guaranteed to be unique and constant
#         for this object during its lifetime.
```

```
#         Two objects with non-overlapping lifetimes may have the same id()␣
 ↪value. This means that as long as an
#         object exists in memory, its id() value will remain the same.

id(c)
```

[40]:
```
140253164671232
```

[41]:
```
d = c
print(id(c) == id(d))          # True - d is the same object as c
print(id(c[0]) == id(d[0]))    # True - d[0] is the same object as c[0]
```

```
True
True
```

[42]:
```
d = copy.copy(c)
print(id(c) == id(d))          # False - d is now a new object
print(id(c[0]) == id(d[0]))    # True - d[0] is the same object as c[0]
```

```
False
True
```

[43]:
```
d = copy.deepcopy(c)
print(id(c) == id(d))          # False - d is now a new object
print(id(c[0]) == id(d[0]))    # False - d[0] is now a new object
```

```
False
False
```

[ ]:
```

```

[44]:
```
######################################################################
xs = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
ys = xs  # simple assignment.
```

[45]:
```
print(xs)
print(ys)
```

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

[46]:
```
# add a new sublist to the original (xs) and this didn't affect the copy (ys).
xs.append(['xy_0','xy_1'])
print(xs)
print(ys)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9], ['xy_0', 'xy_1']]
[[1, 2, 3], [4, 5, 6], [7, 8, 9], ['xy_0', 'xy_1']]
```

[47]:
```
xs[1][0] = 'X'
xs[3][1] = 'xy_2'
print(xs)
print(ys)
```

```
[[1, 2, 3], ['X', 5, 6], [7, 8, 9], ['xy_0', 'xy_2']]
[[1, 2, 3], ['X', 5, 6], [7, 8, 9], ['xy_0', 'xy_2']]
```

[48]:
```
ys.append(['ys_0','ys_3'])
ys[1][0] = 'Y'
ys[3][1] = 'Y_1'
print(xs)
print(ys)
```

```
[[1, 2, 3], ['Y', 5, 6], [7, 8, 9], ['xy_0', 'Y_1'], ['ys_0', 'ys_3']]
[[1, 2, 3], ['Y', 5, 6], [7, 8, 9], ['xy_0', 'Y_1'], ['ys_0', 'ys_3']]
```

[ ]:
```

```