November 4 notes Nicole White

Copies vs. Aliases

```
>>> x = 5
>>> y = x
>>> x
5
>>> y
5
>>> y = y+1
>>> y
6
>>> x                    # y changes, but x does not. This is because y is a copy of x
5
```

BUT:

```
>>> p = Point(3,4)
>>> q = p
>>> q.y = q.y+1
>>> q
Point(3,5)
>>> p                    # The change made to q is visible in p. q is an ALIAS of p
Point(3,5)
```

In memory:
x = 5 sets aside a piece of memory called x containing the value 5
y = x sets aside a piece of memory called y, also containing the value 5 (x)
These are two *different* allocations of memory, and thus y is a COPY of x

p = Point(3,4) sets aside a piece of memory called p that contains an address which *directs* us to the Point object
q = p sets aside a piece of memory called q, also containing an address makes a reference to the SAME piece of memory as p
These refer to the *same* allocation of memory, and thus are simply different names for the same thing (ALIASES)

What if you want to make a copy instead of an alias?

```
>>> p = Point(3,4)
>>> q = Point(p.x, p.y)
```

p.x and p.y are both pieces of VALUE DATA, and so this creates copies (as in the x and y example)

as a function:

```
def copy_point(p):
    return Point(p.x, p.y)
```

RECALL: the LineSeg class

```
class LineSeg:
    def __init__(self,p1,p2):
        self.p1 = p1
        self.p2 = p2
```

We now know that self.p1 and self.p2 are aliases for the p1 and p2 used to initialize the line segment object - this means that when p1 or p2 changes, the line segment also changes

If we want the line segment to remain the same even when the Point changes, we must initialize the LineSeg class with COPIES of the Points rather than aliases

```
class LineSeg:
    def __init__(self,p1,p2):
        self.p1 = copy_point(p1)
        self.p2 = copy_point(p2)
```

Shallow and Deep copying:

A shallow copy of an object makes a copy of all its attributes:

```
>>> seg1 = LineSeg(Point(1,2),Point(3,4))
>>> seg2 = LineSeg(seg1.p1, seg1.p2)      # where seg1.p1 and seg1.p2 are copied as
                                           # reference objects
```

What kinds of changes can be made to one seg that will not be reflected in the other?

- seg2 is not quite an alias of seg1, but p1 and p2 are reference data
- if the Points that make up seg1 are changed, seg2 is changed (because these are aliases of seg1 values)
- However, if seg1 *replaces* p1 or p2 with a new Point, the change is not reflected in seg2 (seg2 still makes reference to the original value)

A deep copy of an object recursively copies all of an object's attributes until value data is reached:

```
>>> seg3 = LineSeg(Point(seg1.p1.x, seg1.p1.y), Point(seg1.p2.x,seg1.p2.y))
```

And seg3 looks like a complete copy of seg1 - no changes to seg1 will be visible in seg3

function to deep-copy line segments:

```
def deep_copy_lineseg(seg):
        return LineSeg(copy_point(seg1.p1), copy_point(seg1.p2))
```

Deep copies are more thorough than shallow copies and thus take longer to complete

Copy of a List object:

```
def shallow_copy(L):
        return List(L.hd, shallow_copy(L.tl))

def deep_copy_int_list(L):
        if L == Empty:
                return Empty
        else:
                return List(L.hd, deep_copy_int_list(L.tl))
```