Intro to Classes & Objects

CIS 211

Recall ... Scopes in Python

```
x = 42

def foo():
    x = 23

foo()
print(x)
```

What does this print?

More important: Why?

And also ...

```
def bar(k):
    m = k
    if k < 0:
        return
    bar(k -1)
    print(m)

bar(5)
```

What does it print?
What does that tell us about the scope of variables?

Another scope

Each *module* (source file) defines a scope
("global" really means module scope)

Each *function activation* defines a scope on the stack

We can create additional *object scopes*

```
class Point:
    """An x,y coordinate pair on the Cartesian plane"""

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def move(self, dx, dy):
        return Point(self.x+dx, self.y+dy)

    def __repr__(self):
        return "Point({},{})".format(self.x, self.y)


p = Point(10,12)
r = p.move(5,5)
print(p)
print(r)
```

```
class Point:
    """An x,y coordinate pair on the Cartesian plane"""

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def move(self, dx, dy):
        return Point(self.x+dx, self.y+dy)

    def __repr__(self):
        return "Point({},{})".format(self.x, self.y)

p = Point(10,12)
r = p.move(5,5)
print(p)
print(r)
```

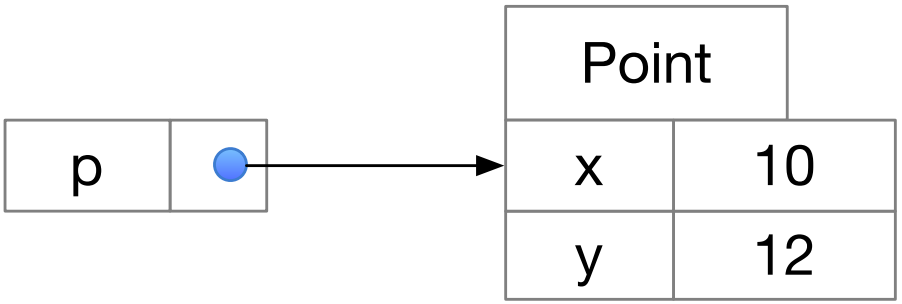
Class definition for Point objects

A point object

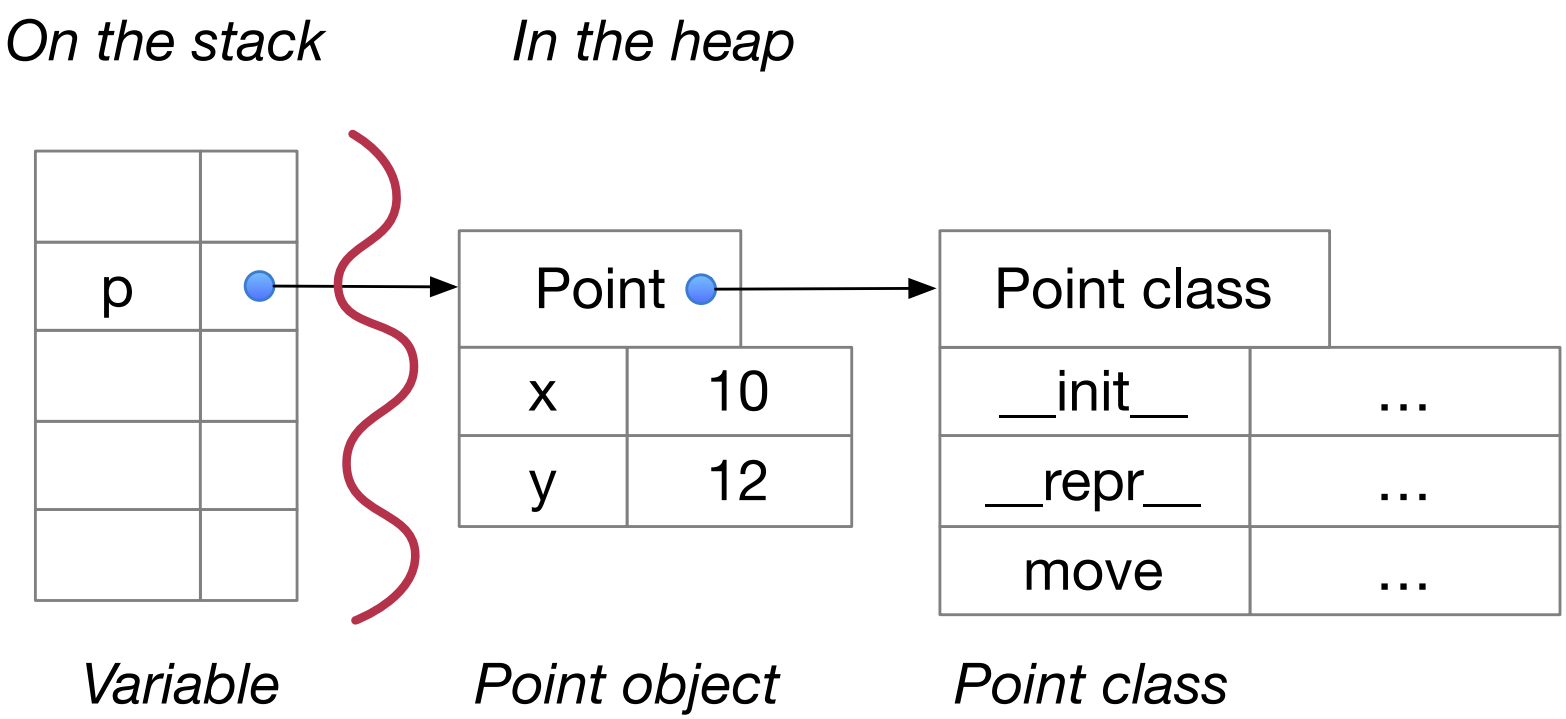
Another point object

```
class Point:
    """An x,y coordinate pair on the Cartesian plane"""

    def __init__(self, x, y):
        self.x = x
        self.y = y

    ...

p = Point(10,12)
```

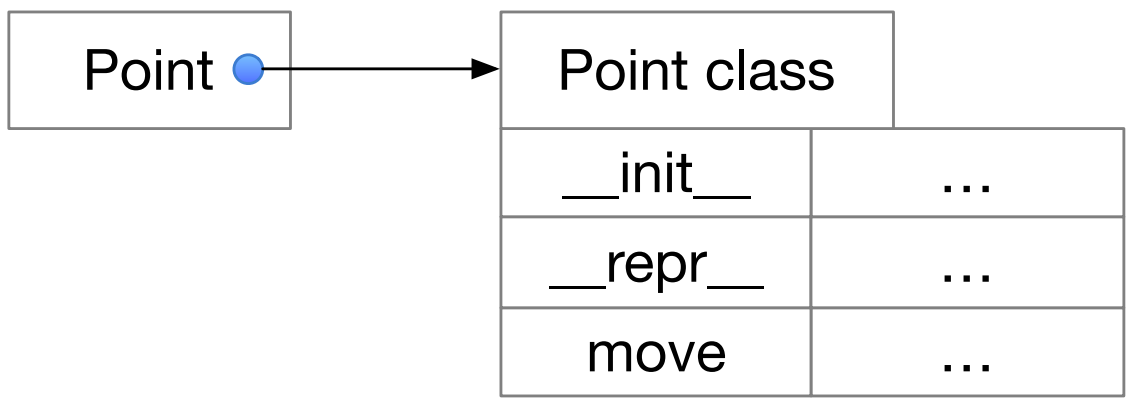
__init__ is the constructor called by Point(10,12)



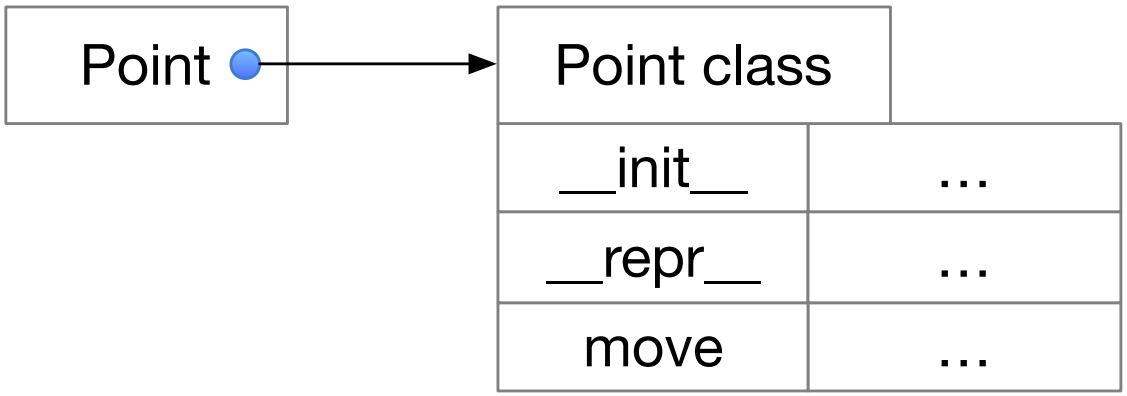
```
...
p = Point(10,12)
...
```

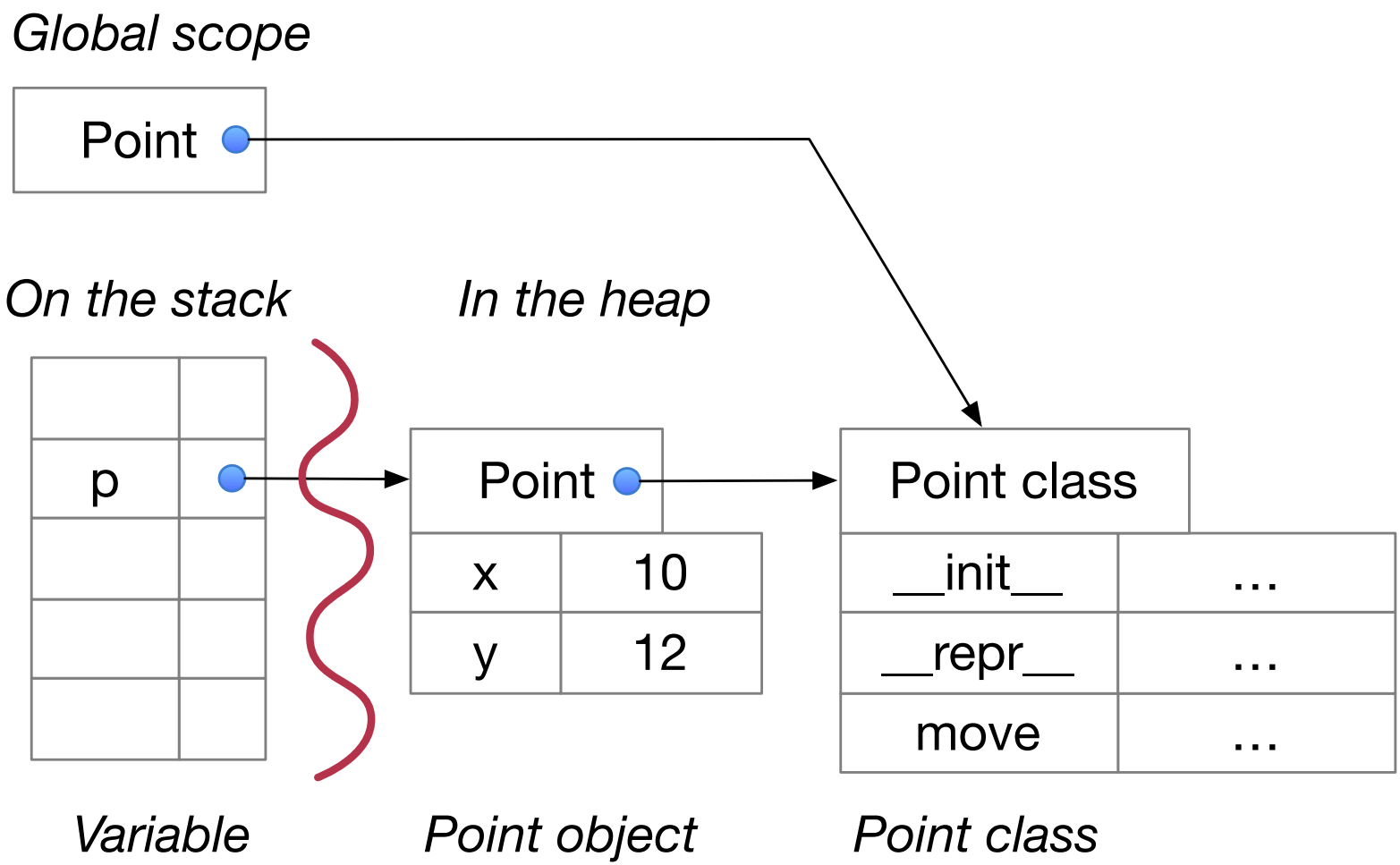

On the stack
In the heap
p
Point
x
10
y
12
Point class
__init__
...
__repr__
...
move
...
Variable
Point object
Point class

```
class Point:
    """An x,y coordinate pair on the Cartesian plane"""

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def move(self, dx, dy):
        return Point(self.x+dx, self.y+dy)

    def __repr__(self):
        return "Point({},{})".format(self.x, self.y)
```

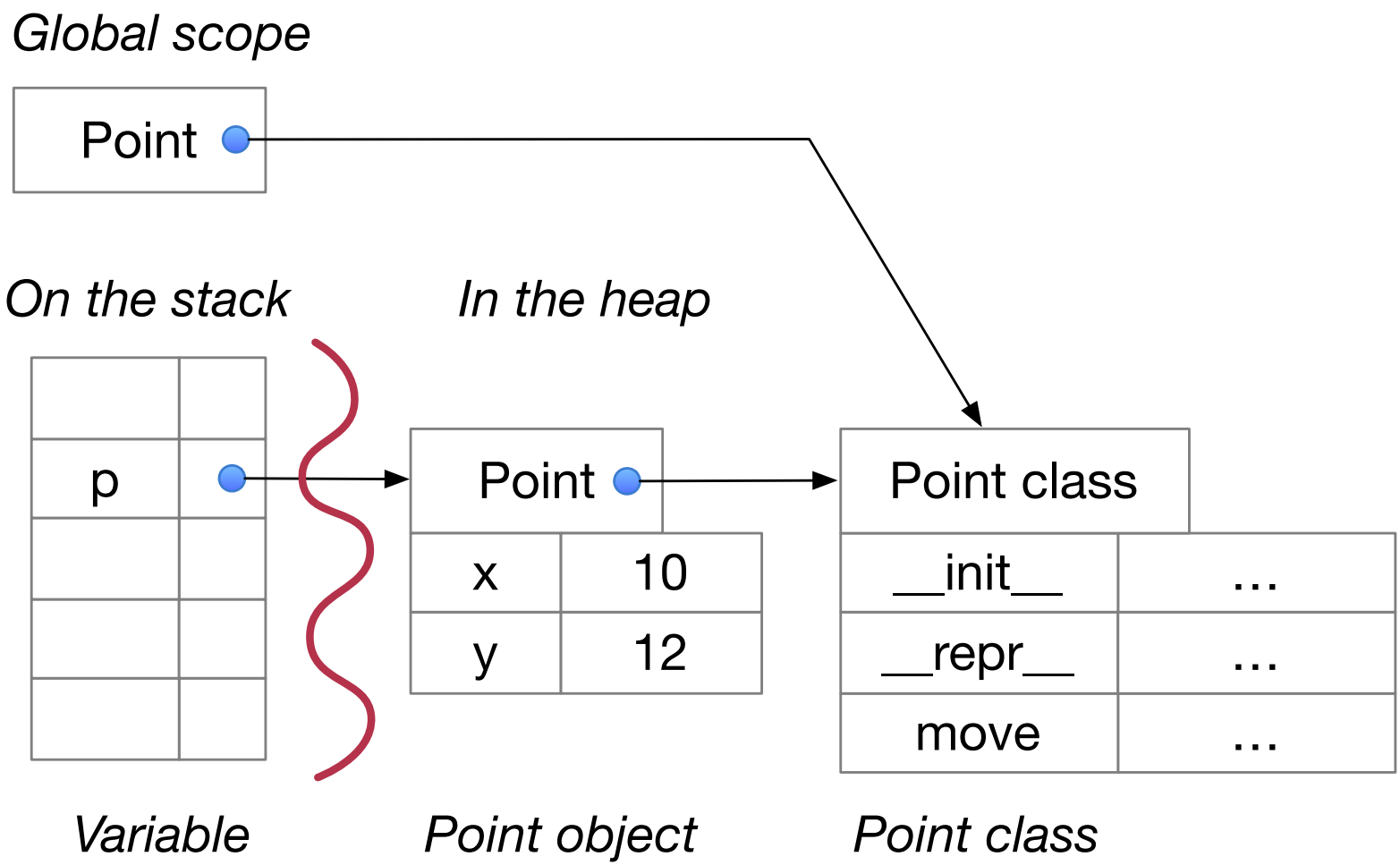


```
class Point:
    """An x,y coordinate pair on the Cartesian plane"""

    def __init__(self, x, y):
        self.x = x
        self.y = y

...

p = Point(10,12)
```




Global scope
Point
On the stack
In the heap
p
Point
x
10
y
12
Point class
__init__
...
__repr__
...
move
...
Variable
Point object
Point class

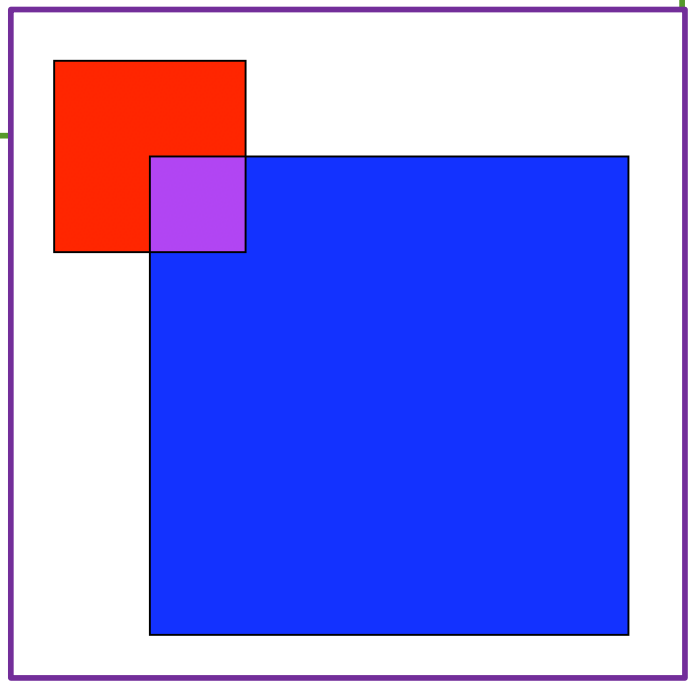
What is p.x?
What is p.move(5,8)?



Your project due Wednesday 8pm

(Yes, this week)

```
$ python3 rect_draw.py
usage: Draw rectangles [-h] ll_x1 ll_y1 ur_x1 ur_y1 ll_x2 ll_y2 ur_x2 ur_y2
Draw rectangles: error: the following arguments are required: ll_x1, ll_y1, ur_x1, ur_y1,
ll_x2, ll_y2, ur_x2, ur_y2

$ python3 rect_draw.py 100 100 200 200 150 150 400 400
Intersection: Rect(Point(150,150),Point(200,200))
```



Calculates the intersection (overlapping region) of two rectangles defined by lower left and upper right corners, (llx,lly) to (urx,ury)

Lots of starter code ... just one method to write!

If time allows:

Our pictures are upside down (lower-left is actually upper-left, upper right is actually lower right) because screen coordinates go from top to bottom.

Where is the best place to fix that?

In the Point object?
In draw_rect?
Elsewhere?
WHY? What do you consider when deciding where to make that change?

(at end of class if time allows)

(Move to demo / *live* coding:
Rectangle overlap)

Point and Rect are classes of immutable objects

Two common styles:

```
p = Point(10,12)
p.move(5,3)
```

```
def move(self, x, y):
    self.x += x
    self.y += y
    return None
```

vs.

```
p = Point(10,12)
p2 = p.move(5,3)
```

```
def move(self, x, y):
    return Point(
        self.x + x,
        self.y + y)
```

Advantages of mutable objects?

Mutable style:

```
p = Point(10,12)
p.move(5,3)
```

```
def move(self, x, y):
    self.x += x
    self.y += y
    return None
```

vs.

```
p = Point(10,12)
p2 = p.move(5,3)
```

```
def move(self, x, y):
    return Point(
        self.x + x,
        self.y + y)
```

Advantages of immutable objects?

Mutable style:

```
p = Point(10,12)
p.move(5,3)
```

```
def move(self, x, y):
    self.x += x
    self.y += y
    return None
```

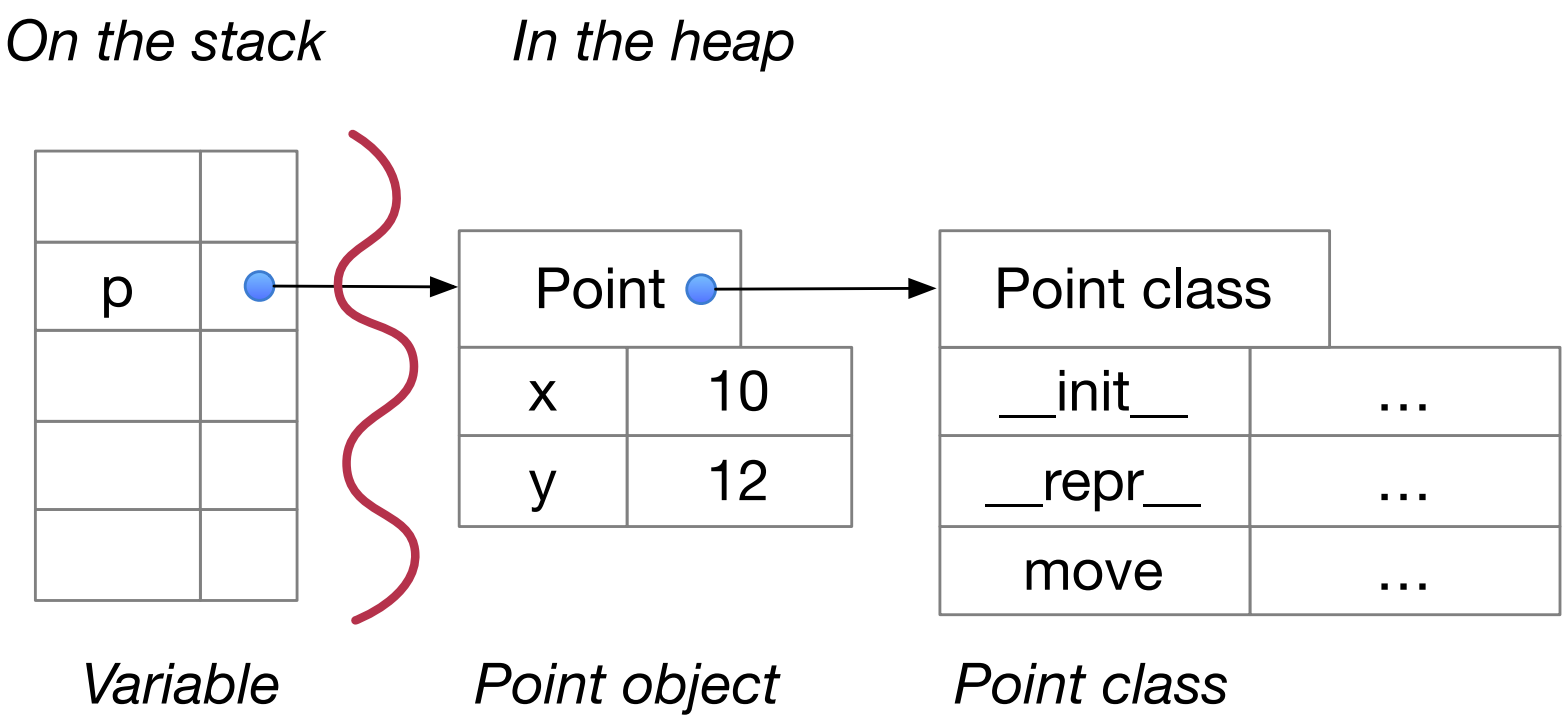
Immutable style

```
p = Point(10,12)
p2 = p.move(5,3)
```

```
def move(self, x, y):
    return Point(
        self.x + x,
        self.y + y)
```

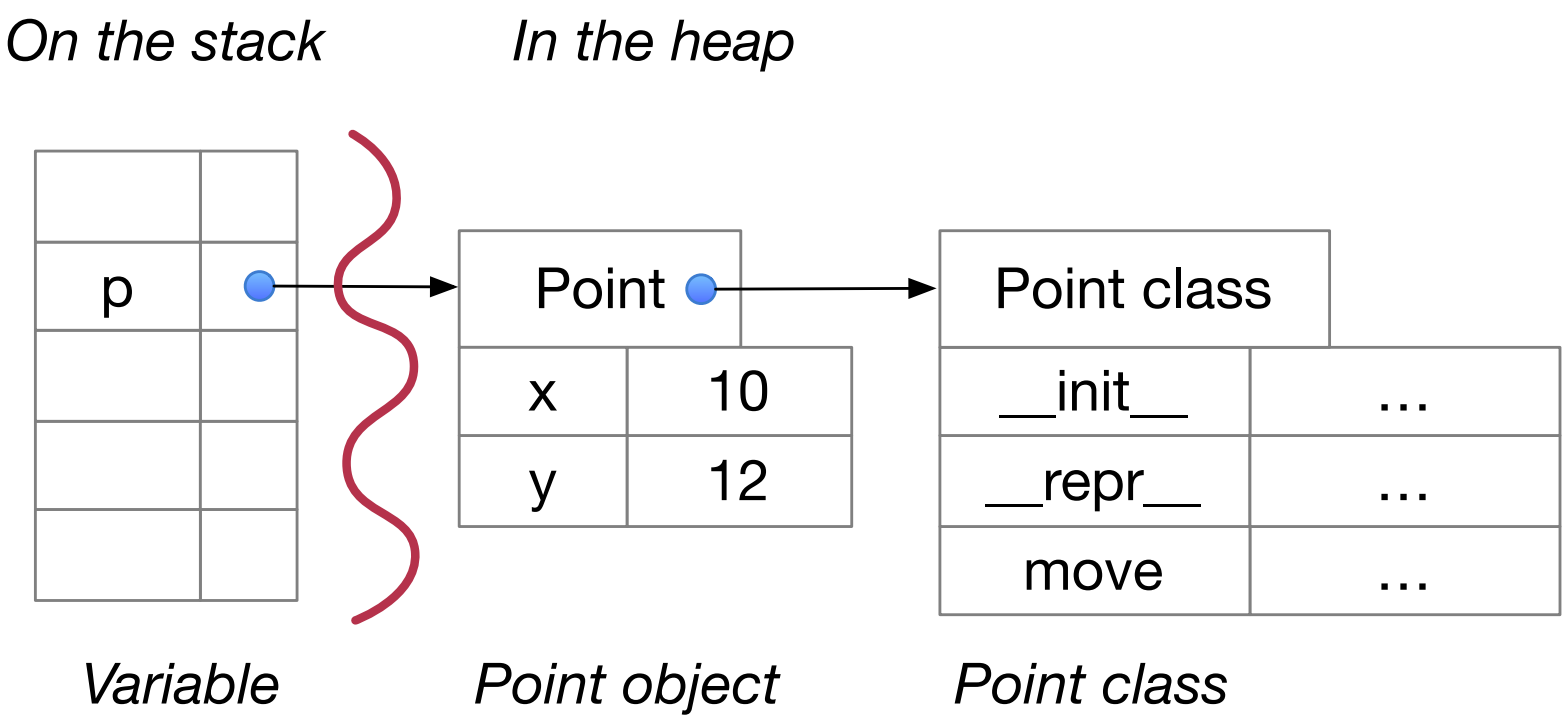
```
...
p = Point(10,12)
...
```

Thinking about mutability: Consider how objects are actually represented



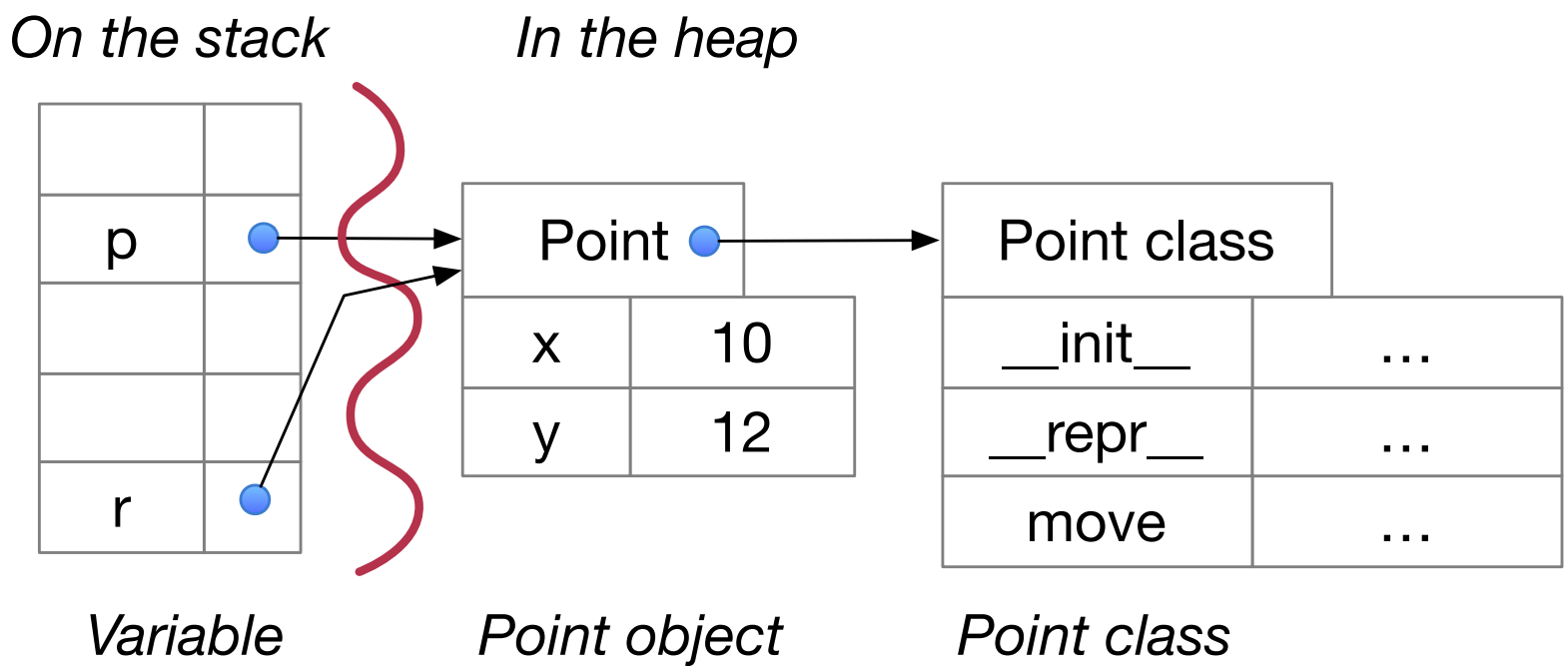
```
...
p = Point(10,12)
...
```

Thinking about mutability: Consider how objects are actually represented



```
...
p = Point(10,12)
...
r = p
r.move(3,4)
```

Immutability may be better with aliasing



If time allows:

Our pictures are upside down (lower-left is actually upper-left, upper right is actually lower right) because screen coordinates go from top to bottom.

Where is the best place to fix that?

In the Point object?
In draw_rect?
Elsewhere?
WHY? What do you consider when deciding where to make that change?

(at end of class if time allows)