

Lecture 14, 6 October 2026

Class `Point` using (x, y) coordinates

```
In [1]: class Point:
        def __init__(self, a=0, b=0): # Constructor
            self.x = a
            self.y = b

        def translate(self, deltax, deltay):
            self.x += deltax # Short for self.x = self.x + deltax
            self.y += deltay

        def odistance(self):
            import math
            d = math.sqrt(self.x*self.x +
                           self.y*self.y)

            return(d)

        def __str__(self):
            return('(' + str(self.x) + ', '
                    + str(self.y) + ')')
```

```
In [2]: p = Point(3,4)
        q = Point(7,10)
        r = Point(3,4)
        s = Point(7,11)
        t = Point()
```

```
In [3]: p
```

```
Out[3]: <__main__.Point at 0x7f6651354980>
```

```
In [4]: print(p)
```

```
(3,4)
```

```
In [5]: str(p)
```

```
Out[5]: '(3,4)'
```

```
In [6]: print(t)
```

```
(0,0)
```

```
In [7]: p.odistance()
```

```
Out[7]: 5.0
```

```
In [8]: q.translate(-1, -2)
```

```
In [9]: str(q)
```

```
Out[9]: '(6,8)'
```

```
In [10]: l1 = [1,2,3]
         l2 = [1,2,3]
         l1 == l2
```

Out[10]: True

```
In [11]: l1 is l2
```

Out[11]: False

```
In [12]: p == r
```

Out[12]: False

```
In [13]: p <= q
```

```
-----
-
TypeError                                Traceback (most recent call las
t)
Cell In[13], line 1
----> 1 p <= q

TypeError: '<=' not supported between instances of 'Point' and 'Point'
```

```
In [14]: class Point:
         def __init__(self,a=0,b=0):
             self.x = a
             self.y = b

         def translate(self,deltax,deltay):
             self.x += deltax
             self.y += deltay

         def odistance(self):
             import math
             d = math.sqrt(self.x*self.x +
                           self.y*self.y)
             return(d)

         def __str__(self):
             return('(' +str(self.x)+', '
                   +str(self.y)+')')

         def __add__(self,p):      # Create a new Point(x1+x2,y1+y2)
             return(Point(self.x + p.x,      # Short for newp = Point(..), return (
                           self.y + p.y))

         def __lt__(self,p):
             return(self.x < p.x and self.y < p.y)

         def __le__(self,p):
             return(self.x <= p.x and self.y <= p.y)

         def __eq__(self,p):
             return(self.x == p.x and self.y == p.y)

         def __ne__(self,p):
             return(not(self == p))
```

```
def __gt__(self,p):
    return(self.x > p.x and self.y > p.y)

def __ge__(self,p):
    return(self.x >= p.x and self.y >= p.y)
```

```
In [15]: p = Point(3,4)
         q = Point(7,10)
         r = Point(3,4)
         s = Point(7,11)
```

```
In [16]: p < q, q < p, p == q, p == r, p <= p, q <= s, s >= q, q > p
```

```
Out[16]: (True, False, False, True, True, True, True, True)
```

```
In [17]: str(p+s)
```

```
Out[17]: '(10,15)'
```

```
In [18]: str(p+s+r)
```

```
Out[18]: '(13,19)'
```

```
In [19]: str(p-s)
```

```
-----
-
TypeError                                Traceback (most recent call las
t)
Cell In[19], line 1
----> 1 str(p-s)

TypeError: unsupported operand type(s) for -: 'Point' and 'Point'
```

```
In [20]: p is r
```

```
Out[20]: False
```

```
In [21]: t = p
```

```
In [22]: t.translate(3,4)
```

```
In [23]: print(p)
```

```
(6,8)
```

```
In [24]: p is t
```

```
Out[24]: True
```

Changing the implementation

- Change the definition of Point to use polar representation, (r, θ)

```
In [25]: import math
         class Point:
```

```

def __init__(self,a=0,b=0):
    self.r = math.sqrt(a*a + b*b)
    if a == 0:
        if b >= 0:
            self.theta = math.pi/2
        else:
            self.theta = 3*math.pi/2
    else:
        self.theta = math.atan(b/a)

def translate(self,deltax,deltay):
    x = self.r*math.cos(self.theta)
    y = self.r*math.sin(self.theta)
    x += deltax
    y += deltay
    self.r = math.sqrt(x*x + y*y)
    if x == 0:
        if y >= 0:
            self.theta = math.pi/2
        else:
            self.theta = 3*math.pi/2
    else:
        self.theta = math.atan(y/x)

def odistance(self):
    return(self.r)

def __str__(self):
    x = self.r*math.cos(self.theta)
    y = self.r*math.sin(self.theta)
    return('(' +str(x)+' , '+str(y)+' )')

def __add__(self,p):
    sx = self.r*math.cos(self.theta)
    sy = self.r*math.sin(self.theta)
    px = p.r*math.cos(p.theta)
    py = p.r*math.sin(p.theta)
    return(Point(sx + px,sy+py))

def __lt__(self,p):
    sx = self.r*math.cos(self.theta)
    sy = self.r*math.sin(self.theta)
    px = p.r*math.cos(p.theta)
    py = p.r*math.sin(p.theta)
    return(sx < px and sy < py)

```

- The interface still assumes (x, y) representation
- When constructing a point, convert (x, y) to (r, θ)
 - Be careful about the case where $x = 0$
- To translate a point, convert (r, θ) back to (x, y) , translate, then convert back to (r, θ)
- Similar conversion for `__str__()`, `__add__()`, `__lt__()`

Repeat the examples above

- Observe that nothing changes for the user of the class

```
In [26]: p = Point(3,4)
        q = Point(7,10)
```

```
In [27]: p.odistance(), q.odistance()
```

```
Out[27]: (5.0, 12.206555615733702)
```

```
In [28]: p.translate(3,4)
        p.odistance()
```

```
Out[28]: 10.0
```

```
In [29]: print(p) # Note some lack of precision going from (x,y) to (r,theta) and
        (6.0000000000000001,7.999999999999999)
```

```
In [30]: str(p)
```

```
Out[30]: '(6.0000000000000001,7.999999999999999)'
```

```
In [31]: print(p+q)
        (13.0000000000000002,18.0)
```

```
In [32]: print(p,q)
        (6.0000000000000001,7.999999999999999) (6.999999999999999,10.0)
```

```
In [33]: p < q, q < p
```

```
Out[33]: (True, False)
```

A note about variables inside classes

- Without the prefix `self`, variables are internal to a function
- Variables with prefix `self` persist within the object

```
In [34]: class Experiment:
        def __init__(self,a):
            x = a

        def __str__(self):
            return(str(x))
```

```
In [35]: z = Experiment(5)
```

```
In [36]: str(z)
```

```

-----
NameError                                Traceback (most recent call last)
Cell In[36], line 1
----> 1 str(z)

Cell In[34], line 6, in Experiment.__str__(self)
      5     def __str__(self):
----> 6         return(str(x))

NameError: name 'x' is not defined

```

```

In [37]: class Experiment2:
        def __init__(self,a):
            self.x = a

        def __str__(self):
            return(str(self.x))

```

```

In [38]: y = Experiment2(7)
        str(y)

```

Out[38]: '7'

- The name `self` for the current object (first parameter) is only a convention
- Can use any other name

```

In [39]: class Experiment3:
        def __init__(self,a):
            self.x = a

        def __str__(this):
            return(str(this.x))

```

```

In [40]: x = Experiment3(17)
        print(x)

```

17

- Classes and objects were grafted onto Python as an afterthought
- If we have a class `C`, and object `o` and a function `f(self,x,y,z)` inside `C`, we can replace `o.f(a,b,c)` by `C.f(o,a,b,c)`.
- In other words, `self` is actually a reference to the object on which `f` is being invoked.

```

In [41]: pnew = Point(5,7)
        pnew.odistance()

```

Out[41]: 8.602325267042627

```

In [42]: Point.odistance(pnew)

```

Out[42]: 8.602325267042627

```
In [43]: Point.translate(pnew,4,7)
```

```
In [44]: print(pnew)
```

```
(9.0,14.0)
```

- This is also true for built in datatypes like `list`

```
In [45]: l = [1,2,3]
l.append(4)
```

```
In [46]: l
```

```
Out[46]: [1, 2, 3, 4]
```

```
In [47]: list.append(l,5)
```

```
In [48]: l
```

```
Out[48]: [1, 2, 3, 4, 5]
```

- Python also has no mechanism to ensure privacy of implementation
- We cannot prevent code outside the class from accessing internal fields `p.x` and `p.y` for a `Point p`
- In fact, we can even add new internal fields!

```
In [49]: pnew.z = 7
```

Linked lists

- An implementation using classes and objects; compare with our earlier implementation using nested dictionaries
- An empty list has a single node with `value` and `next` both `None`
- Last node in the list has `next` set to `None`

```
In [50]: class List:
    def __init__(self):
        self.value = None
        self.next = None
        return

    def isempty(self):
        return(self.value == None)

    def append(self,v):    # append, iterative
        if self.isempty():
            self.value = v
            return

        temp = self
        while temp.next != None:
            temp = temp.next
```

```

temp.next = List()
temp.next.value = v
return

def insert(self,v):
    if self.isempty():
        self.value = v
        return

    newnode = List()
    newnode.value = v

    # Exchange values in self and newnode
    (self.value, newnode.value) = (newnode.value, self.value)

    # Switch links
    (self.next, newnode.next) = (newnode, self.next)

    return

def __str__(self):
    # Iteratively create a Python list from linked list
    # and convert that to a string
    selflist = []
    if self.isempty():
        return(str(selflist))

    temp = self
    selflist.append(temp.value)

    while temp.next != None:
        temp = temp.next
        selflist.append(temp.value)

    return(str(selflist))

```