Lecture 11, 19 September 2023

Defining our own data structures

- In Lecture 8, we implemented a "linked" list using dictionaries
- The fundamental functions like `listappend`, `listinsert`, `listdelete` modify the underlying list
- Instead of `mylist = {}`, we wrote `mylist = createlist()`
- To check empty list, use a function `isempty()` rather than `mylist == {}`
- Can we clearly separate the **interface** from the **implementation**
- Define the data structure in a more "modular" way

Object oriented approach

- Describe a datatype using a template, called a **class**
- Create independent instances of a class, each is an **object**
- Each object has its own internal *state* -- the values of its local variables
- All objects in a class share the same functions to query/update their state
- `l.append(x)` vs `append(l,x)`
  - Tell an object what to do vs passing an object to a function
- Each object has a way to refer to itself

Basic definition of class `Point` using (x, y) coordinates

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

  def translate(self,deltax,deltay):
    self.x += deltax  # Same as self.x = self.x deltax
    # In general, if we have a = a op b for any arithmetic operation op, can write a op= b
    # For example: a += 5 is a = a + 5, a -= 10 is a = a - 10 etc
    self.y += deltay

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

Create two points

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

Compute `odistance` for `p` and `q`

In [3]:
```
p.odistance(), q.odistance()
```

Out[3]:
```
(5.0, 12.206555615733702)
```

Translate `p` and check the distance

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

Out[4]:
```
10.0
```

- At this stage, `print()` does not produce anything meaningful
- `+` is not defined yet

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

```
<__main__.Point object at 0x7f9c7a1c9090>
```

In [6]:
```
print(p+q)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [6], line 1
----> 1 print(p+q)

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

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

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

In [7]:
```
import math
class Point:
  def __init__(self,a,b):
    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)
```

Repeat the examples above

- Observe that nothing changes for the user of the class

Create two points

In [8]:
```
p = Point(3,4)
q = Point(7,10)
```

Compute `odistance` for `p` and `q`

In [9]:
```
p.odistance(), q.odistance()
```

Out[9]:
```
(5.0, 12.206555615733702)
```

Translate `p` and check the distance

In [10]:
```
p.translate(3,4)
p.odistance()
```

Out[10]:
```
10.0
```

- We have not yet defined special functions `__str__`, `__add__` etc

In [11]:
```
print(p)
```

```
<__main__.Point object at 0x7f9c686bdff0>
```

In [12]:
```
print(p+q)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [12], line 1
----> 1 print(p+q)

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

Return to (x, y) representation, adding `__str__` and `__add__`

In [13]:
```
class Point:
  def __init__(self,a,b):
    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):
    return(Point(self.x + p.x,
                 self.y + p.y))
  # Previous line is a concise way of saying
  #
  # newx = self.x + p.x
  # newy = self.y + p.y
  # newpt = Point(newx,newy)
  # return(newpt)
```

Again, run the same examples

In [14]:
```
p = Point(3,4)
q = Point(7,10)
```

Compute `odistance` for `p` and `q`

In [15]:
```
p.odistance(), q.odistance()
```

Out[15]:
```
(5.0, 12.206555615733702)
```

Translate `p` and check the distance

In [16]:
```
p.translate(3,4)
p.odistance()
```

Out[16]:
```
10.0
```

In the following two cells, we see a difference

- Since `__str__` is defined, `print()` gives useful output
- `+` works as expected thanks to the definition for `__add__`

In [17]:
```
print(p)
```

```
(6,8)
```

In [18]:
```
str(p)
```

Out[18]:
```
'(6,8)'
```

In [19]:
```
print(p+q)
```

```
(13,18)
```

In [20]:
```
print(p,q)
```

```
(6,8) (7,10)
```

- Still no function `__lt__` so `<` is not defined

In [21]:
```
p < q
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [21], line 1
----> 1 p < q

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

- Define `__lt__` to enable `p < q`

In [22]:
```
class Point:
  def __init__(self,a,b):
    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):
    return(Point(self.x + p.x,
                 self.y + p.y))
  # Previous line is a concise way of saying
  #
  # newx = self.x + p.x
  # newy = self.y + p.y
  # newpt = Point(newx,newy)
  # return(newpt)

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

In [23]:
```
p = Point(3,4)
q = Point(7,10)
```

In [24]:
```
p < q, q < p
```

Out[24]:
```
(True, False)
```

- Object oriented style in Python is a "hack"
- Can call a function defined inside a class by passing the object as the first parameter
- Use the class name to tell Python where to find the function

In [25]:
```
Point.translate(p,9,10)
```

In [26]:
```
print(p)
```

```
(12,14)
```

- This also works for built-in types like lists

In [27]:
```
l = [1,2,3]
```

In [28]:
```
list.append(l,4)
```

In [29]:
```
l
```

Out[29]:
```
[1, 2, 3, 4]
```

- Though the implementation of an object should be hidden, Python has no way to enforce this
- Can access internal fields from outside

In [30]:
```
p.x, p.y
```

Out[30]:
```
(12, 14)
```

- May also want to define auxiliary functions for internal use that should not be visible outside
- In (r, θ) version of `Point`, code to convert `(x,y)` to `(r,theta)` is duplicated in `__init__` and `translate`
- Could define an auxiliary function `convert(x,y)` that returns corresponding `(r,theta)`
- But no mechanism in Python to ensure that `convert()` is available only internally -- external user can also call `convert()`

A note about variables inside classes

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

In [31]:
```
class Experiment:
    def __init__(self,a):
        x = a

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

In [32]:
```
z = Experiment(5)
```

In [33]:
```
str(z)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [33], line 1
----> 1 str(z)

Cell In [31], line 6, in Experiment.__str__(self)
      5 def __str__(self):
----> 6     return(str(x))

NameError: name 'x' is not defined
```

In [34]:
```
class Experiment2:
    def __init__(self,a):
        self.x = a

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

In [35]:
```
y = Experiment2(7)
```

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

Out[36]:
```
'7'
```

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

In [37]:
```
class Experiment3:
    def __init__(self,a):
        self.x = a

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

In [38]:
```
x = Experiment3(17)
```

In [39]:
```
print(x)
```

```
17
```