Classes and objects

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming and Data Structures with Python

Lecture 10

Classes and objects

- Abstract datatype
  - Stores some information
  - Designated functions to manipulate the information
  - For instance, list: `append()`, `insert()`, `delete()`, ...

How? Not specified

Classes and objects

- Abstract datatype
  - Stores some information
  - Designated functions to manipulate the information
  - For instance, list: `append()`, `insert()`, `delete()`, ...
- Separate the (private) implementation from the (public) specification

Classes and objects

- Abstract datatype
  - Stores some information
  - Designated functions to manipulate the information
  - For instance, list: `append()`, `insert()`, `delete()`, ...
- Separate the (private) implementation from the (public) specification
- Class
  - Template for a data type
  - How data is stored
  - How public functions manipulate data

Classes and objects

- Abstract datatype
  - Stores some information
  - Designated functions to manipulate the information
  - For instance, list: `append()`, `insert()`, `delete()`, ...
- Separate the (private) implementation from the (public) specification
- Class
  - Template for a data type
  - How data is stored
  - How public functions manipulate data
- Object
  - Concrete instance of template

Example: 2D points

- A point has coordinates (x, y)
  - `__init__()` initializes internal values `x`, `y`
  - First parameter is always `self`

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

constructor

l = createlist()

p = Point(6,4)

Create a new instance of Point and assign to p

Example: 2D points

- A point has coordinates (x, y)
  - `__init__()` initializes internal values `x`, `y`
  - First parameter is always `self`
- Translation: shift a point by $(\Delta x, \Delta y)$
  - $(x, y) \mapsto (x + \Delta x, y + \Delta y)$

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

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

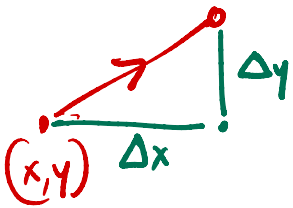
"My" x = a

"My" y = b

self.x = self.x + deltax

l.append(v)

append(l,v)



translate(p, deltax, deltay)

p.translate(deltax, deltay)

Example: 2D points

- A point has coordinates (x, y)
  - `__init__()` initializes internal values `x`, `y`
  - First parameter is always `self`
- Translation: shift a point by $(\Delta x, \Delta y)$
  - $(x, y) \mapsto (x + \Delta x, y + \Delta y)$
- Distance from the origin
  - $d = \sqrt{x^2 + y^2}$

$\sqrt{(x-0)^2 + (y-0)^2}$

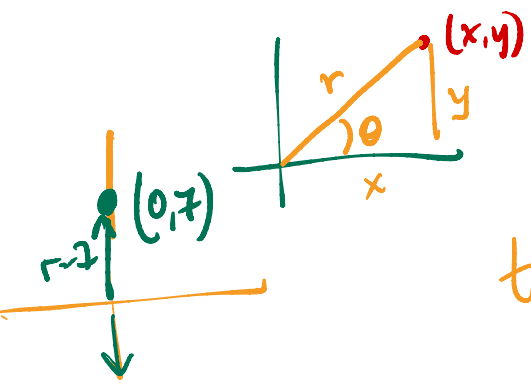
$(0,0) \longrightarrow (x,y)$

```
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)
```

Polar coordinates

- (r, θ) instead of (x, y)
  - $r = \sqrt{x^2 + y^2}$
  - $\theta = \tan^{-1}(y/x)$

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



$\tan\theta = \frac{y}{x}$

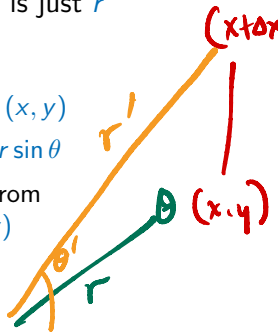
Polar coordinates

- (r, θ) instead of (x, y)
  - $r = \sqrt{x^2 + y^2}$
  - $\theta = \tan^{-1}(y/x)$
- Distance from origin is just r

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

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

Polar coordinates

- (r, θ) instead of (x, y)
  - $r = \sqrt{x^2 + y^2}$
  - $\theta = \tan^{-1}(y/x)$
- Distance from origin is just r
- Translation
  - Convert (r, θ) to (x, y)
  - $x = r\cos\theta$, $y = r\sin\theta$
  - Recompute r, θ from $(x + \Delta x, y + \Delta y)$




```
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:
    self.theta = math.pi/2
  else:
    self.theta = math.atan(y/x)
```

Polar coordinates

- (r, θ) instead of (x, y)
  - $r = \sqrt{x^2 + y^2}$
  - $\theta = \tan^{-1}(y/x)$
- Distance from origin is just r
- Translation
  - Convert (r, θ) to (x, y)
  - $x = r\cos\theta$, $y = r\sin\theta$
  - Recompute r, θ from $(x + \Delta x, y + \Delta y)$
- Interface has not changed
  - User need not be aware whether representation is (x, y) or (r, θ)

```
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:
    self.theta = math.pi/2
  else:
    self.theta = math.atan(y/x)
```

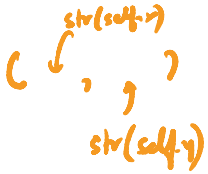
Special functions

- `__init__()` — constructor

Special functions

- `__init__()` — constructor
- `__str__()` — convert object to string
  - `str(o) == o.__str__()`
  - Implicitly invoked by `print()`

```
class Point:

  ...

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



Special functions

- `__init__()` — constructor
- `__str__()` — convert object to string
  - `str(o) == o.__str()__`
  - Implicitly invoked by `print()`
- `__add__()`
  - Implicitly invoked by `+`

```
class Point:
  ...

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

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

p.add(q)
q.add(p)

$p - (x_p, y_p)$

$q - (x_q, y_q)$

$r = p+q \rightsquigarrow (x_p+x_q, y_p+y_q)$

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

```
    newx = self.x + p.x
    newy = self.y + p.y
    newpt = Point(newx,
                  newy)
    return(newpt)
```

Special functions

- `__init__()` — constructor
- `__str__()` — convert object to string
  - `str(o) == o.__str()__`
  - Implicitly invoked by `print()`
- `__add__()`
  - Implicitly invoked by `+`
- Similarly
  - `__mult__()` invoked by `*`
  - `__lt__()` invoked by `<`
  - `__ge__()` invoked by `>=`
  - ...

```
class Point:

  ...

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


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