Classes and objects

Madhavan Mukund

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

Programming, Data Structures and Algorithms using Python

Week 1

Classes and objects

- Abstract datatype
  - Stores some information
  - Designated functions to manipulate the information
  - For instance, stack: last-in, first-out, `push()`, `pop()`

Classes and objects

- Abstract datatype
  - Stores some information
  - Designated functions to manipulate the information
  - For instance, stack: last-in, first-out, `push()`, `pop()`
- Separate the (private) implementation from the (public) specification

Classes and objects

- Abstract datatype
  - Stores some information
  - Designated functions to manipulate the information
  - For instance, stack: last-in, first-out, `push()`, `pop()`
- 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, stack: last-in, first-out, `push()`, `pop()`
- 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`
  - Here, by default a point is at $(0, 0)$

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

Example: 2D points

- A point has coordinates (x, y)
  - `__init__()` initializes internal values `x`, `y`
  - First parameter is always `self`
  - Here, by default a point is at $(0, 0)$
- Translation: shift a point by $(\Delta x, \Delta y)$
  - $(x, y) \mapsto (x + \Delta x, y + \Delta y)$

```
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
```

Example: 2D points

- A point has coordinates (x, y)
  - `__init__()` initializes internal values `x`, `y`
  - First parameter is always `self`
  - Here, by default a point is at $(0, 0)$
- Translation: shift a point by $(\Delta x, \Delta y)$
  - $(x, y) \mapsto (x + \Delta x, y + \Delta y)$
- Distance from the origin
  - $d = \sqrt{x^2 + y^2}$

```
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)
```

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

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=0,b=0):
    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))
```

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))
```