ES 112
Introduction to Objected Oriented Programming
Nov 11, 2015

Computing

IIT Gandhinagar, India

Nov, 2015

Classes

- A class is a user defined type
- Suppose we want to do some geometry in 2D. We need to define points

```
class Point(object):
    """Represents a point in 2-D space."""
```

- You have a data type called Point

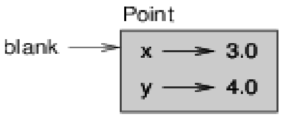
```
>>> blank = Point()
>>> print blank
<__main__.Point instance at 0xb7e9d3ac>
```

- We can now already use this to store more attributes

```
>>> blank.x = 3.0
>>> blank.y = 4.0
```

Attributes

- We first created a class (Point) and then an object from the class (blank)
- Once you have assigned attributes of an object, you can access them and change them
  - Objects are mutable



```
 >>> print blank.y
4.0
>>> blank.x = 3.0
```

  - Can pass to functions

```
def print_point(p):
    print '(%f, %f)' % (p.x, p.y)
```

Exercise

Write a function called `distance_between_points` that takes two `Points` as arguments and returns the distance between them.

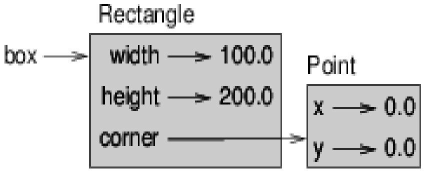
More classes

- Suppose now, we want to create a class for a rectangle
- Should have a
  - Length, width and the left hand corner point

-
  ```
  class Rectangle(object):
      """Represents a rectangle.

      attributes: width, height, corner.
      """
  ```

```
box = Rectangle()
box.width = 100.0
box.height = 200.0
box.corner = Point()
box.corner.x = 0.0
box.corner.y = 0.0
```



Exercise

- Objects can be return values:

- Write a function find_center that takes a Rectangle as an argument and returns a Point that contains the coordinates of the center of the Rectangle

Mutability

- Objects are mutable

- Write a function named move_rectangle that takes a Rectangle and two numbers named dx and dy. It should change the location of the rectangle by adding dx to the x coordinate of corner and adding dy to the y coordinate of corner.

History

- An `object` is a collection of data and methods grouped together.

History

- An `object` is a collection of data and methods grouped together.
- A `class` is a collection of objects of similar type.

History

- An `object` is a collection of data and methods grouped together.
- A `class` is a collection of objects of similar type.
- Previously, the *logic* to manipulate the data was the main concern of a programmer.

History

- An `object` is a collection of data and methods grouped together.
- A `class` is a collection of objects of similar type.
- Previously, the *logic* to manipulate the data was the main concern of a programmer.
- With growing length of programs, programmers felt the need to group similar type of data and the functions (methods) that act on the data.

History

- An `object` is a collection of data and methods grouped together.
- A `class` is a collection of objects of similar type.
- Previously, the *logic* to manipulate the data was the main concern of a programmer.
- With growing length of programs, programmers felt the need to group similar type of data and the functions (methods) that act on the data.
- Smalltalk was the first language to introduce object oriented programming.

History

- An `object` is a collection of data and methods grouped together.
- A `class` is a collection of objects of similar type.
- Previously, the *logic* to manipulate the data was the main concern of a programmer.
- With growing length of programs, programmers felt the need to group similar type of data and the functions (methods) that act on the data.
- Smalltalk was the first language to introduce object oriented programming.
- With Java it got hugely popular.

Example: List Object

- List has elements (data).

Example: List Object

- List has elements (data).
- List has a collection of methods (functions) that can act on the list (e.g., `append()`, `pop()`, `remove()` etc.).

Example: List Object

- List has elements (data).
- List has a collection of methods (functions) that can act on the list (e.g., `append()`, `pop()`, `remove()` etc.).
- In Python everything (lists,numbers,functions,..) is an object.

Example: Circle

- Previously we defined a class and then attached the attributes. Usually this is `NOT` what is done in practice.

Example: Circle

- Previously we defined a class and then attached the attributes. Usually this is `NOT` what is done in practice.
- The attributes are decided first and kept in the class. Extra attributes are rarely added.

Example: Circle

- Previously we defined a class and then attached the attributes. Usually this is `NOT` what is done in practice.
- The attributes are decided first and kept in the class. Extra attributes are rarely added.
- We will create an object called circle.

Example: Circle

- Previously we defined a class and then attached the attributes. Usually this is NOT what is done in practice.
- The attributes are decided first and kept in the class. Extra attributes are rarely added.
- We will create an object called circle.
- Identify the data attributes: origin of the circle (x,y), and radius.

```
#class definition: This creates a "Class Object"
class Circle():
    x=0
    y=0
    r=3
    #a method to compute area
    #a method to compute circumference

print Circle.r
#instantiation
c=Circle()
print c.r
```

Example: Circle

- Previously we defined a class and then attached the attributes. Usually this is NOT what is done in practice.
- The attributes are decided first and kept in the class. Extra attributes are rarely added.
- We will create an object called circle.
- Identify the data attributes: origin of the circle (x,y), and radius.
- Identify the functions associated with a circle: Find area, find circumference.

```python
#class definition: This creates a "Class Object"
class Circle():
    x=0
    y=0
    r=3
    #a method to compute area
    #a method to compute circumference

print Circle.r
#instantiation
c=Circle()
print c.r
```

Setting Parameters During Instantiation

```
#class definition: This creates a "Class Object"
class Circle():
    x=0
    y=0
    r=3
    def __init__(self,x1,y1,r1):
        self.x=x1
        self.y=y1
        self.r=r1
    #a method to compute area
    #a method to compute circumference


#instantiation
c=Circle(1,1,5) #this makes an instance of the Circle class,

print c.r
```

- The method `__init__()` is called during instantiation.
- Any method defined inside a class definition has `self` as it's first argument.
- It refers to the same object.

Add Other Methods

```
#class definition: This creates a "Class Object"
class Circle():
    x=0
    y=0
    r=3
    def __init__(self,x1,y1,r1):
        self.x=x1
        self.y=y1
        self.r=r1
    #a method to compute area
    def area(self):
        return 3.1415*self.r*self.r
    #a method to compute circumference
    def circm(self):
        return 2*3.1415.self.r

#instantiation
c=Circle(1,1,5) #this makes an instance of the Circle class,

print c.r
print c.area() # Argument? c is the argument
```

- The first argument comes before the dot.
- Exercise Add a method to check if a given point (p,q) is inside the circle.
  Hint: `def inside(self, p,q)`