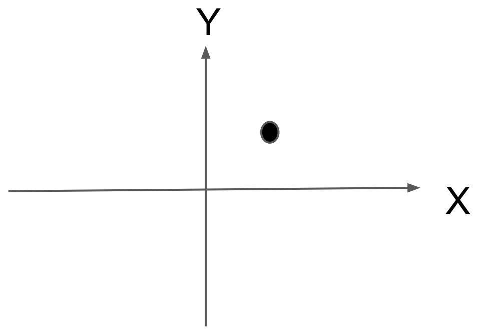
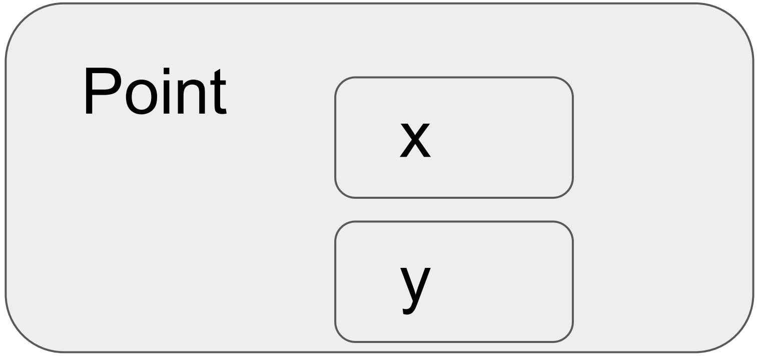
Defining Structured Data (Class/Object)

Winter 2019
@ Cal Poly SLO
By
Toshi

Structured Data using objects

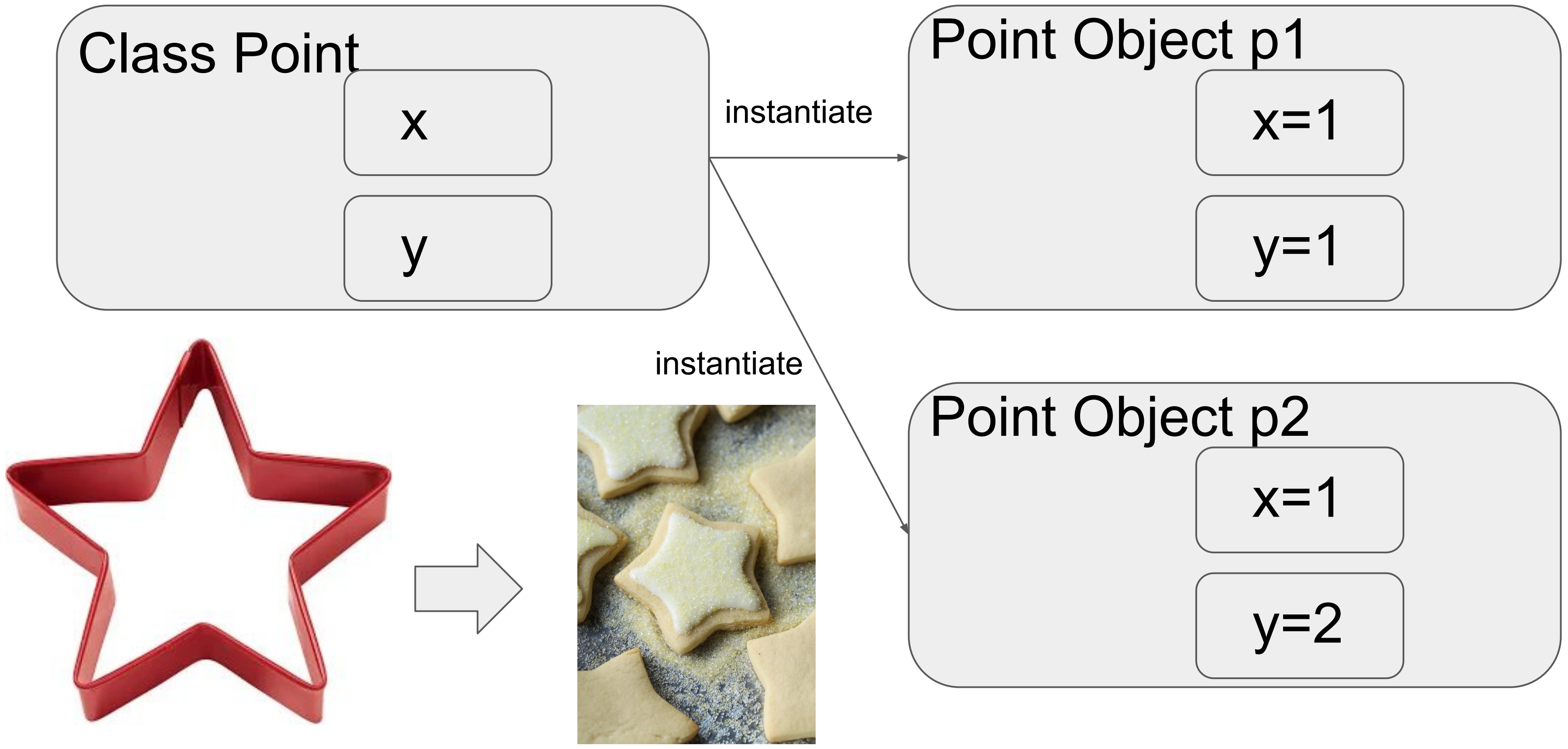
- You can extend the language
  - By defining your custom class (type)
    - E.g, class Point with two integer fields x and y to represent a point in a 2D cartesian coordinate system.





```
point1 = Point(1,1)
```

Class and Object



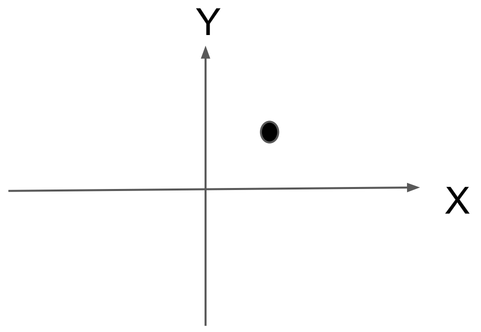
Accessing Fields in an object

- Dot Notation
  - object_name.field_name

```
p1 = Point(1, 1)
print p1.x, p1.y

p2 = Point(2, 3)
print p2.x, p2.y
```

Defining Point class



```
class Point:

    #self denotes this object itself
    def __init__(self, x, y):
        #int value of x coordinate
        self.x = x

        #int value of y coordinate
        self.y = y

'''create a new object of Point class with x=1 and y=1 and assign it to
variable point1'''
point1 = Point(1,1)
```

Built-in Methods in class

- Every Python class has these so called “methods” by default:
    - __init__(self, [...])
        - called when a new object of the class is created. Used for initializing the object.
    - __repr__(self)
        - computes the “official” string representation of an object
    - __eq__(self, other)
        - checks equality with the other object.

__repr__

```
class Point:

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return “Point(%s, %s)”\
           % (self.x, self.y)
```

```
>>> point = Point(1,1)
>>> print point
    Point(1, 1)
```

__eq__

```
class Point:

    def __init__(self, x, y):
        self.x = x
        self.y = y

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

```
>>> point1 = Point(1,1)
>>> point2 = Point(1,2)
>>> point3 = Point(1,2)
>>> print point1 == point2
    False
>>> print point2 == point3
    True
```

Using predefined class methods

```
class Point:

    #self denotes this object itself
    def __init__(self, x, y):
        self.x = x
        self.y = y

    #official string representation
   def __repr__(self):
        return “(%s, %s)” % (self.x, self.y)

    #define equality
    def __eq__(self, other):
        return self.x == other.x and \
            self.y == other.y
```

```
p1 = Point(1,1)
print p1    # (1,1)

p2 = Point(2,3)
print p2    # (2,3)

p3 = Point(1,1)
print p3    # (1,1)

print p1 == p2  # False

print p1 == p3  # True
```

More examples of functions

```
class Point:

    #self denotes this object itself
    def __init__(self, x, y):
        self.x = x
        self.y = y

    #official string representation
   def __repr__(self):
        return “(%s, %s)” % (self.x, self.y)

    #define equality
    def __eq__(self, other):
        return self.x == other.x and \
            self.y == other.
```

```
def distance(p1, p2):
    Return math.sqrt((p1.x - p2.x)**2 +
         (p1.y - p2.y)**2)

p1 = Point(0,0)
print p1    # (1,1)

p2 = Point(2,2)
print p2    # (2,2)

d = distance(p1, p2)

print “distance = ”, d   # distance = 1.41421
```

Using Python Built-in Modules

- Python has some built-in modules.
    - math
        - Provides mathematical functions for computing square root, log, and others.
    - To use a built-in module in your program, you need to import it first.
    - Use dot notation to access methods defined in a built-in module.

```
import math

math.sqrt(2) # square root of 2

math.log(2) # log of 2
```