

CIS 122

A Class of One's Own

Classes

- Custom objects
 - Composed of properties and methods
- Properties store information
 - Coordinates
 - Names
- Methods tell object how to act
 - `__init__`
 - `__repr__`

Classes

p1

xcor \rightarrow 3
ycor \rightarrow 5

p2

xcor \rightarrow 0
ycor \rightarrow 0

p3

xcor \rightarrow 1
ycor \rightarrow 7

Class Methods Under the Surface

- Class methods all start with a special argument
 - Generally named "self"
 - Refers to the object calling the method
- What really happens when we call a class method?
 - What happens to that first argument?

Class Methods Under the Surface

```
class Point:
```

```
    def __init__(self, x, y):  
        <init code>
```

```
    def __repr__(self):  
        return "(" + str(self.xcor) + ", " + str(self.ycor) + ")"
```

```
    def absValue(self):  
        return math.sqrt(self.xcor**2 + self.ycor**2)
```

```
print p
```

Class Methods Under the Surface

```
class Point:
```

```
    def __init__(self, x, y):  
        <init code>
```

```
    def __repr__(self):  
        return "(" + str(self.xcor) + ", " + str(self.ycor) + ")"
```

```
    def absValue(self):  
        return math.sqrt(self.xcor**2 + self.ycor**2)
```

```
print p.__repr__()
```

Class Methods Under the Surface

```
class Point:
```

```
    def __init__(self, x, y):  
        <init code>
```

```
    def __repr__(self):  
        return "(" + str(self.xcor) + ", " + str(self.ycor) + ")"
```

```
    def absValue(self):  
        return math.sqrt(self.xcor**2 + self.ycor**2)
```

```
print p.__repr__()  
print Point.__repr__(p)
```

Class Methods Under the Surface

- When Python calls a class method
 - The object gets substituted in for the first argument

`p.__repr__()` \rightarrow `print Point.__repr__(p)`
`p.absVal()` \rightarrow `Point.absVal(p)`

- The constructor is a little strange
 - But works the same way

Adding up your Points

- How do we add two points?
 - Sum their x coordinates
 - Sum their y coordinates
- For example
 - $(1, 3) + (10, 20) = (11, 23)$
 - $(2, 2) + (-2, -2) = (0, 0)$
 - $(0, 0) + (0, 0) = (0, 0)$

Adding up your Points

- Let's define addition for our Point class
- `__add__` method
 - Defines "+" operator for our class
 - Takes two arguments

```
def __add__(self, other):
```

Adding up your Points

- Let's define addition for our Point class
- `__add__` method
 - Defines "+" operator for our class
 - Takes two arguments

```
def __add__(self, other):  
    newX = self.xcor + other.xcor  
    newY = self.ycor + other.ycor  
    newPoint = Point(newX, newY)  
    return newPoint
```

Comparing Points

- How does Python compare objects?
- Everything boils down to numbers
 - Ints - compare values
 - Floats - compare values
 - Characters - compare ord values
 - Strings - compare characters
- To compare points, we'll need a basis for comparison
 - How would we like to compare two points?

Comparing Points

- Python has special comparison methods
 - `__gt__` → `>`
 - `__ge__` → `>=`
 - `__lt__` → `<`
 - `__le__` → `<=`
 - `__eq__` → `==`
 - `__ne__` → `!=`
- That's a lot of methods to define
 - It would be nice if we could define just one

Comparing Points

- Python has one method covering all comparisons
- `__cmp__(a,b)`
 - Takes two arguments
 - Returns a number
 - Negative if $a < b$
 - Positive if $a > b$
 - 0 if $a == b$
- Let's write a `__cmp__` method for our point class

Get the Point

- We now have a functioning Point class
 - Constructor
 - Representation
 - Distance from origin
 - Addition
 - Comparison
- We could add more functionality
 - Depends on what we're using it for