

classes in classes, functions, and collections.

Week 10 | Lecture 2 (10.2)

if nothing else, write `#cleancode`

This Week's Content

- **Lecture 10.1**
 - objects, classes, and methods
 - Reading: Chapter 14
- **Lecture 10.2**
 - classes in classes, functions, and collections
 - Reading: Chapter 14
- **Lecture 10.3**
 - Design Problem: Robot Localization

OOP Recap

- Everything in Python is an object.

Is **this** an instance
of **this** class.



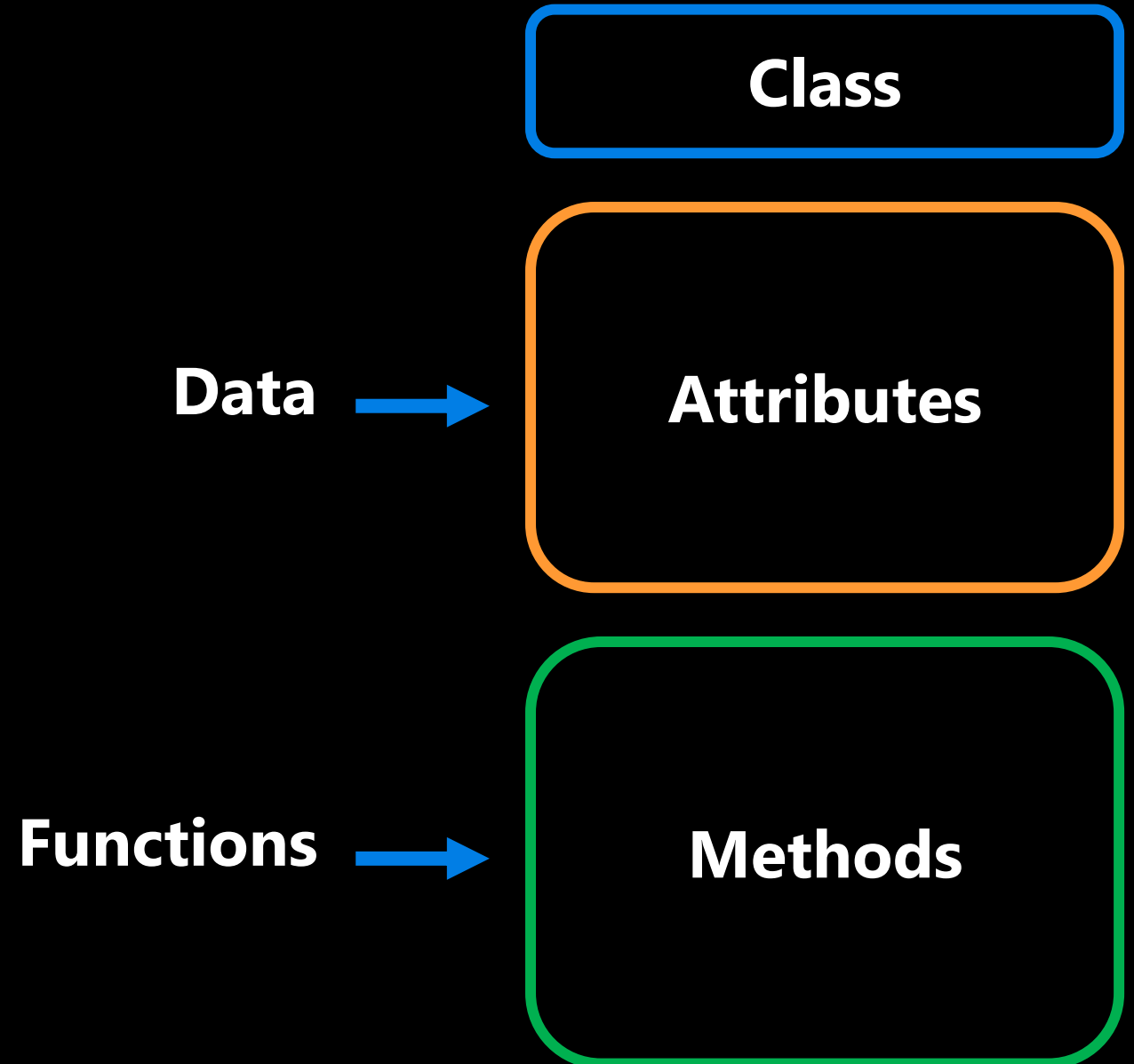
```
>>> isinstance(4, object)  
True
```

```
>>> isinstance(max, object)  
True
```

```
>>> isinstance("Hello", object)  
True
```

OOP Recap

- A class can be thought of as a template for the objects that are instances of it.



OOP Recap

Instances
(objects) of the
Turtle class.



name: Lucy
x location: 24
y location: 35



name: Susmit
x location: 134
y location: 45



name: Brian
x location: 92
y location: 62

Turtle

name
x location
y location

move up
move down
move left
move right
go to

OOP Recap

- General form of a Class:
 - Class Name
 - CamelCase
 - CourseGrades
 - BankAccount
 - FlightStatus
 - XRayImage
 - Constructor
 - Methods

```
class Name:
```

```
    def __init__(self, param1, param2, ...):  
        self.param1 = param1  
        self.param2 = param2
```

```
    ...  
    body
```

```
    def method1(self, parameters):  
        body
```

```
    def method2(self, parameters):  
        body
```

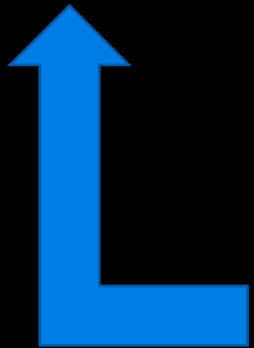
```
    def method3(self, parameters):  
        body
```

OOP Recap

- **Instantiate:** Creating (constructing) an instance of a class.



```
alex = Turtle(0, 0)
```

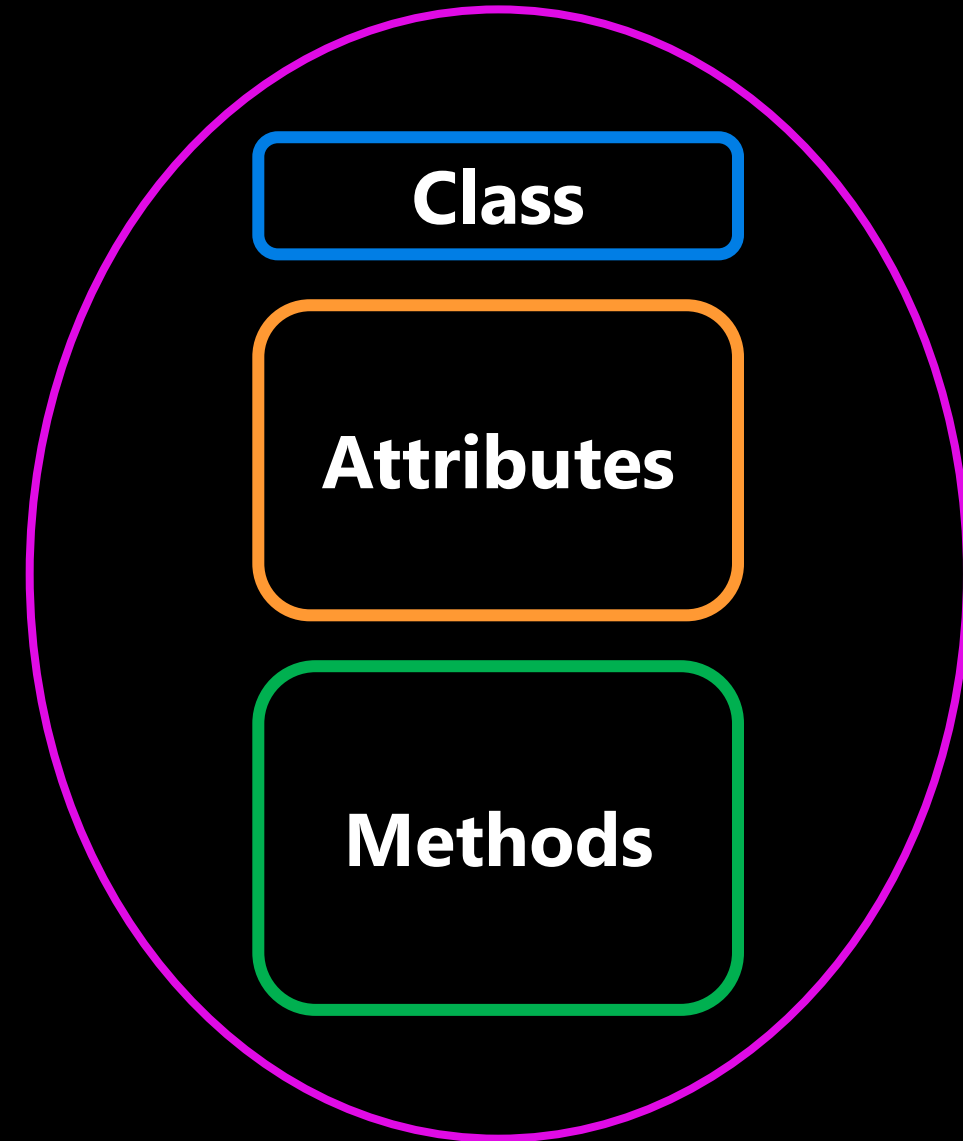


This is the process of instantiating.

OOP Recap

- The core of object-oriented programming is the organization of the program by **encapsulating** related **data** and **functions** together in an object.
- Encapsulation permits objects to operate completely independently of each other as discrete and self-contained bunch of data and code.

Encapsulation



OOP Recap

- **self**
- Reference to the instance of the class.

```
class Turtle:

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

    def up(self):
        self.y += 1

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

    def get_position(self):
        return self.x, self.y
```

OOP Recap

```
katia = Turtle(0, 0)
```

```
class Turtle:

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

    def up(katia):
        katia.y += 1

    def goto(katia, x, y):
        katia.x = x
        katia.y = y

    def get_position(katia):
        return katia.x, katia.y
```

OOP Recap

```
ben = Turtle(0, 0)
```

```
class Turtle:
```

```
    def __init__(ben, x, y):  
        ben.x = x  
        ben.y = y
```

```
    def up(ben):  
        ben.y += 1
```

```
    def goto(ben, x, y):  
        ben.x = x  
        ben.y = y
```

```
    def get_position(ben):  
        return ben.x, ben.y
```

OOP Recap

```
seb = Turtle(0, 0)
```

```
class Turtle:
```

```
    def __init__(seb, x, y):  
        seb.x = x  
        seb.y = y
```

```
    def up(seb):  
        seb.y += 1
```

```
    def goto(seb, x, y):  
        seb.x = x  
        seb.y = y
```

```
    def get_position(seb):  
        return seb.x, seb.y
```

OOP Recap

- Because at the time of designing the class we don't know what these instance names will be, we just chose one.
 - **self**

```
class Turtle:

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

    def up(self):
        self.y += 1

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

    def get_position(self):
        return self.x, self.y
```

OOP Recap

- Although you do not technically need to use the word **self**, it is widely adopted and is recommended.
 - **this**
 - **instance**
 - **thing**
 - **self** ← Python Standard

```
class Turtle:

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

    def up(self):
        self.y += 1

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

    def get_position(self):
        return self.x, self.y
```

OOP Recap

- Accessing attributes (Data) and methods (Functions) is different.

```
ben = Turtle(0, 0)
```

```
ben.x ← Attribute.
```

```
ben.up() ← A Method is a function, and  
we call functions using  
parentheses.
```

```
def my_func():  
    print("Hello")
```

my_func This function has
not been called.

OOP Recap

```
seb = Turtle(0, 0)
```



These parameters are
passed to the
constructor (the
`__init__` method).

```
class Turtle:
```

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

```
    def up(self):  
        self.y += 1
```

```
    def goto(self, x, y):  
        self.x = x  
        self.y = y
```

```
    def get_position(self):  
        return self.x, self.y
```


Review **Point** Class

- Our Point class from last lecture:
 - Contain data about the location of a Point instance.
 - Be able to calculate the distance between the Point instance and another point.
 - Be able to calculate distance between the Point and the origin.

Point

x
y

distance between points
distance from the origin

Add new method



Review **Point** Class

- Our Point class from last lecture:
 - Contain data about the location of a Point instance.
 - Be able to calculate the distance between the Point instance and another point.
 - Be able to calculate distance between the Point and the origin.

**Open your
notebook**

Click Link:

1. Point Class

Methods and **self**


Inside Class Definition

```
class Turtle:

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

    def goto(self, x, y):
        body
        self.print_location()

    def print_location(self):
        body
```



self refers to the instance
inside the class definition.

Outside Class Definition

```
alex = Turtle(10, 12)

alex.print_location()
```



The instance variable
name refers to the
instance outside the
class definition.

Methods **and** **self**

Inside Class Definition

```
class Turtle:

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

    def goto(self, x, y):
        body
        self.print_location()

    def print_location(self):
        body
```

When defining a method, the first parameter refers to the instance being manipulated (**self**).

Outside Class Definition

```
alex = Turtle(10, 12)

alex.print_location()
```

Methods **and** **self**


Inside Class Definition

```
class Turtle:

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

    def goto(x, y):
        body
        self.print_location()

    def print_location():
        body
```



A common error is to omit the **self** argument as the first parameter of a method definition.

Outside Class Definition

```
alex = Turtle(10, 12)

alex.print_location()
```

Methods **and** **self**

Inside Class Definition

```
class Turtle:

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

    def goto(x, y):
        body
        self.print_location()

    def print_location():
        body
```

TypeError:

`print_location()`
takes 0 positional
arguments but 1
was given.

A method call
automatically
inserts an instance
reference as the
first argument.

Outside Class Definition

```
alex = Turtle(10, 12)

alex.print_location()
```



The error only occurs
when you call the
function.

Defining the methods
without **self** will not
cause an error.

Methods **and** **self**

Inside Class Definition

```
class Turtle:

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

    def goto(self, x, y):
        body
        self.print_location()

    def print_location(self):
        body
```

A method call automatically inserts an instance reference as the first argument.

Outside Class Definition

```
alex = Turtle(10, 12)
alex.print_location()
```

Python does this for us.



The error only occurs when you call the function.

Defining the methods without **self** will not cause an error.

Calling Methods

- There are two ways to call methods.
- **Method 1**
- One way is to access the method through the class name and pass in the object.
 - `Class.method(instance_of_class)`
- **Method 2**
- The other is to use object-oriented syntax.
 - `instance_of_class.method()`

Open your notebook

Click Link:

2. Calling Methods

Objects **and** Functions

- Functions and methods can return instances.
- For example, given two Point objects, what if you want to create a point halfway in between?

```
def calc_midpoint(self, point):
```

body

```
    return Point(x, y)
```

Add new method

Point

x
y

distance between points
distance from the origin
midpoint between points

Objects **and** Functions

- Functions and methods can return instances.
- For example, given two Point objects, what if you want to create a point halfway in between?

```
def calc_midpoint(self, point):  
    body  
    return Point(x, y)
```

**Open your
notebook**

Click Link:

**3. Add Midpoint
Method to Point
Class**

Variable Declarations Are Optional

- While we can assign each point to a variable, is not necessary.

- `p1 = Point(3, 4)`
 - `p2 = Point(5, 12)`
 - `p3 = p1.midpoint(p2)`

- Here is an alternative that uses no explicit variables.

- `p3 = Point(3, 4).halfway(Point(5, 12))`

↑
Instance
of Point

↑
Instance
of Point

↑
Instance
of Point

**Open your
notebook**

Click Link:

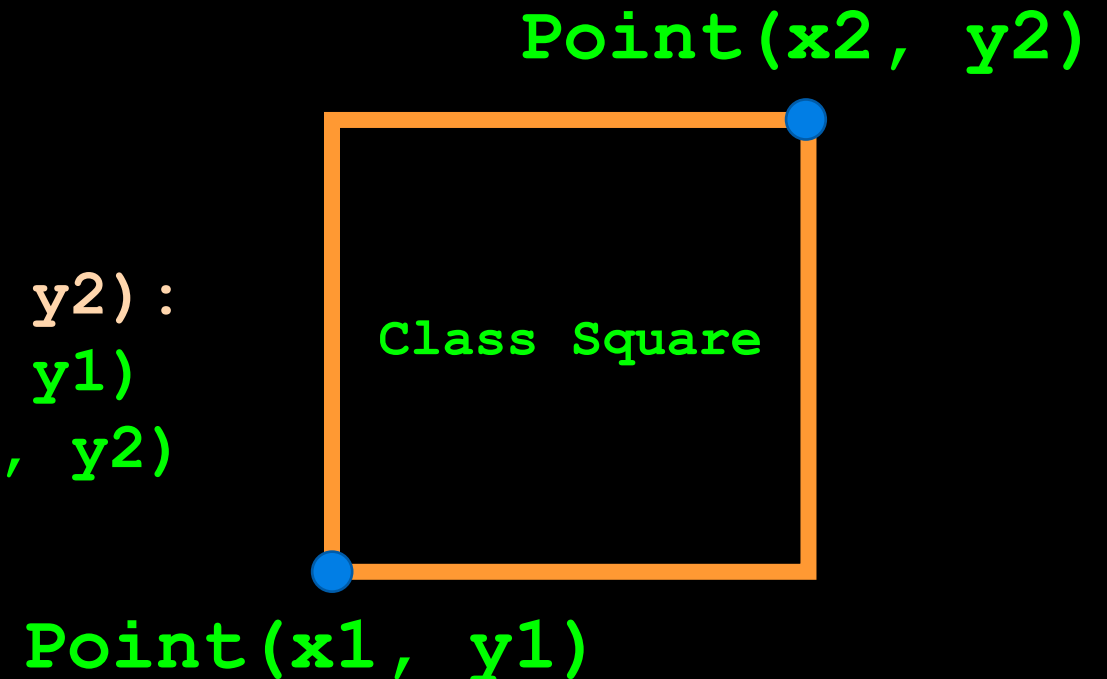
**4. Variable
Declarations Are
Optional**

Objects as Data Attributes of Classes

- Objects are programmer-created data types that can be used just like other data types.
- In particular, the data in an object can be in the form of instances of other classes.

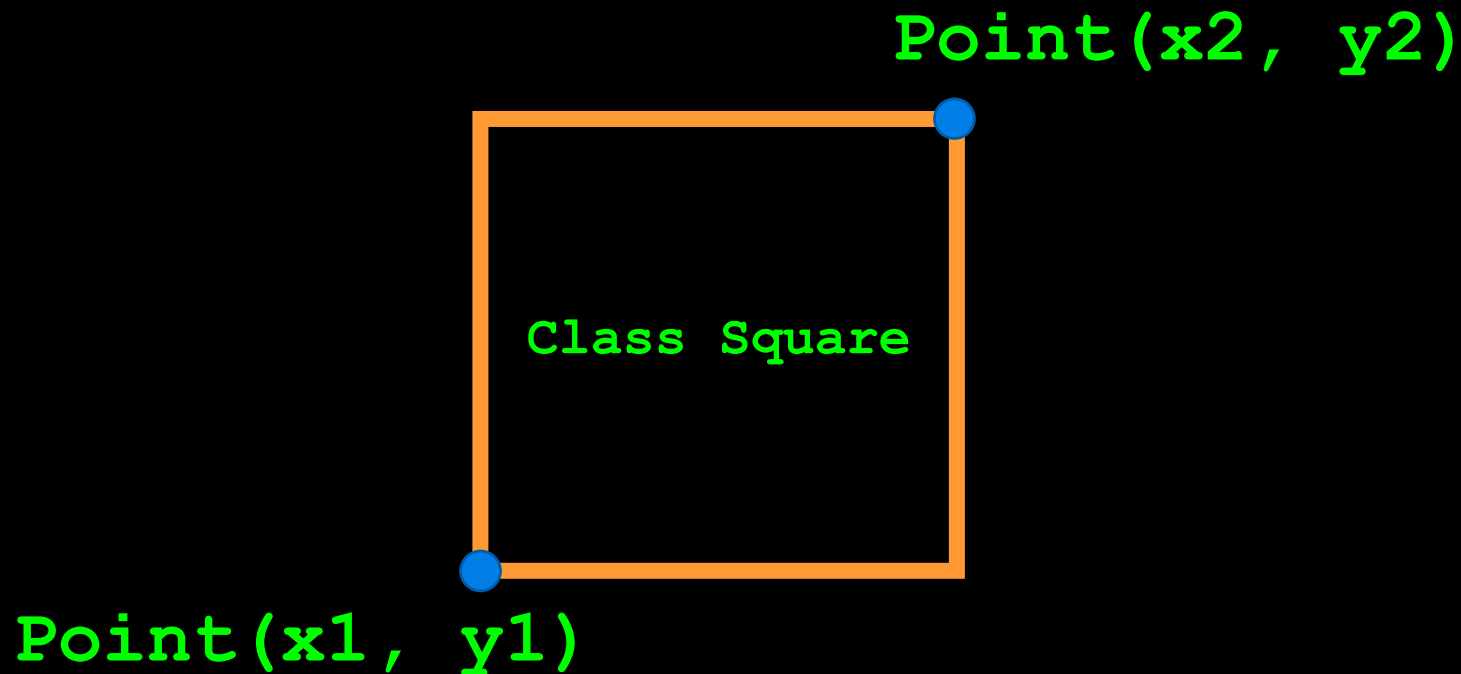
```
class Square:
```

```
    def __init__(self, x1, x2, y1, y2):  
        self.lower_left = Point(x1, y1)  
        self.upper_right = Point(x2, y2)
```



Objects as Data Attributes

- Create a Square class and use Point instances and attributes.



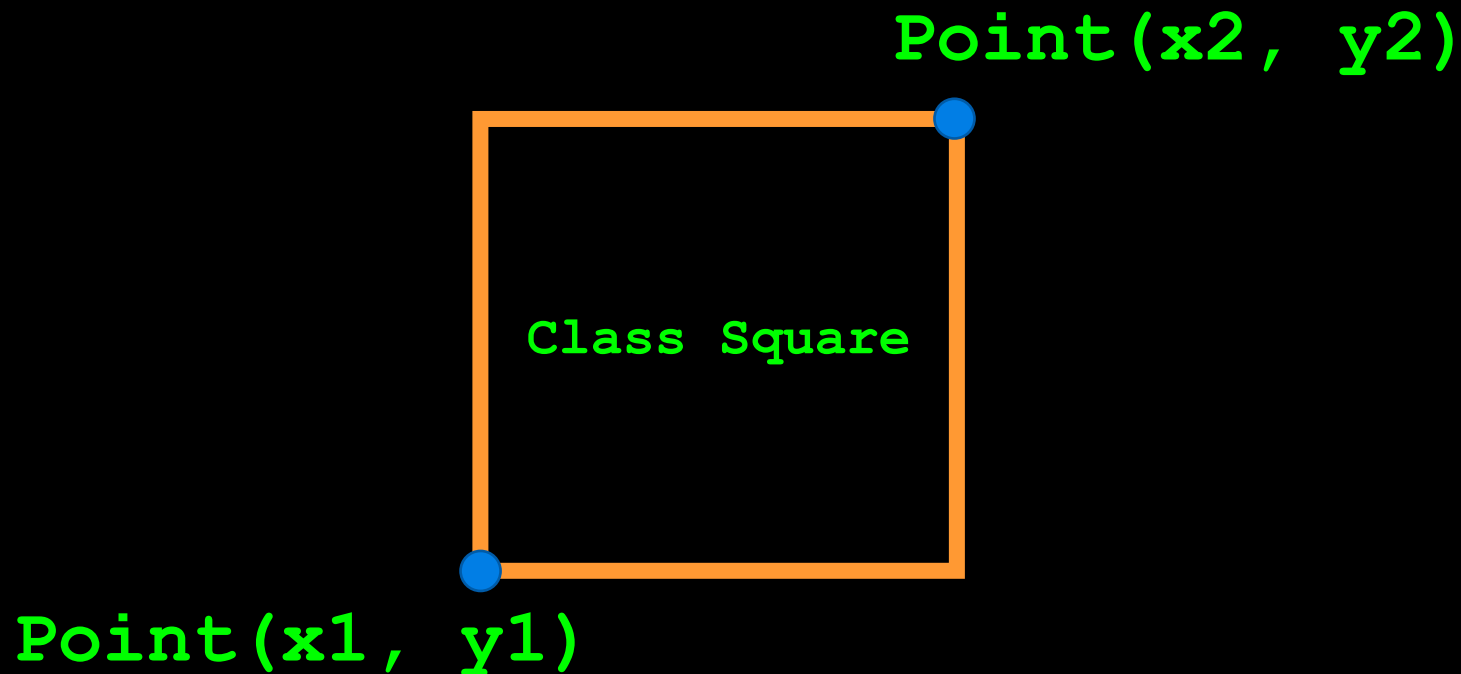
Square

**lower_left
upper_right**

**calculate area
calculate centre**

Objects as Data Attributes of Classes

- Create a Square class and use Point instances and attributes.



Open your notebook

Click Link:

5. Objects as Data Attributes of Classes

Objects In Collections

- Of course, you can put objects in Python collections like **lists**, **tuples**, etc.

**Open your
notebook**

Click Link:
**6. Objects In
Collections**

Printing Attribute Information

- It would be nice to not have to write a `print` statement each time we want to display some attribute information.
 - `p = Point(3, 4)`
 - `print(p.x, p.y)`
- Is there some way we could encapsulate this process?
- It would be better if we could have a method take care of it.

**Open your
notebook**

Click Link:

**7. Printing Attribute
Information**

Patient Class

- What if you are writing a medical application that needs to keep track of patients and their data.
- Let's create a `PatientData` class.
- **Attributes**
 - `height_cm`
 - `weight_kg`
- **Methods**
 - `print_data()`

**Open your
notebook**

Click Link:
8. Patient Class

classes in classes, functions, and collections.

Week 10 | Lecture 2 (10.2)

if nothing else, write `#cleancode`