**6.100L Recitation 9**

**Reminders**

* MQ 9 on Monday 11/21

**Review from the last 2 weeks…**

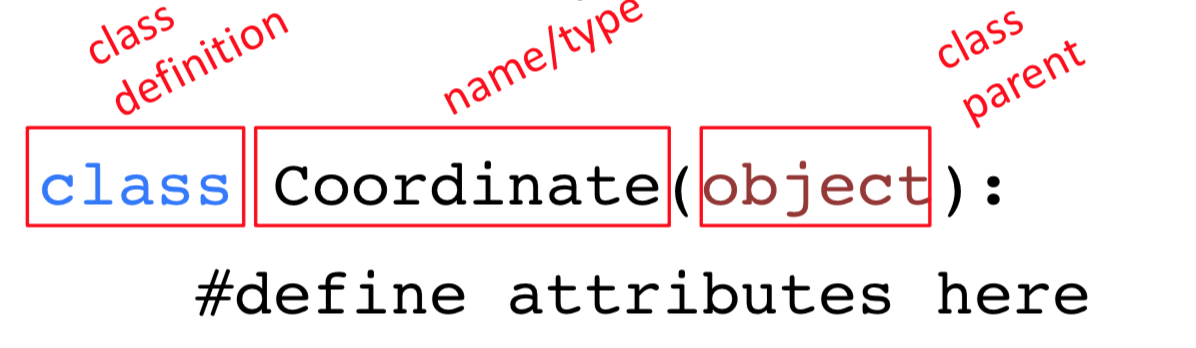
**Object Orientation Programming (OOP)**

* Objects allow you to store data in python
* Everything in python is an object
* Class defines a type of object
  + so far in class we have seen the following built-in classes: int, float, string, list, tuples, dictionaries
* An object is an instance of its class
  + for example: 3, “hello”, [1,2,3] are all instances of a class
* Advantages of OOP:
  + Allows you to bundle data into packages
  + Reduces complexity of your code, making it easy to reuse code
  + Allows you to implement & test behaviour of each class separately

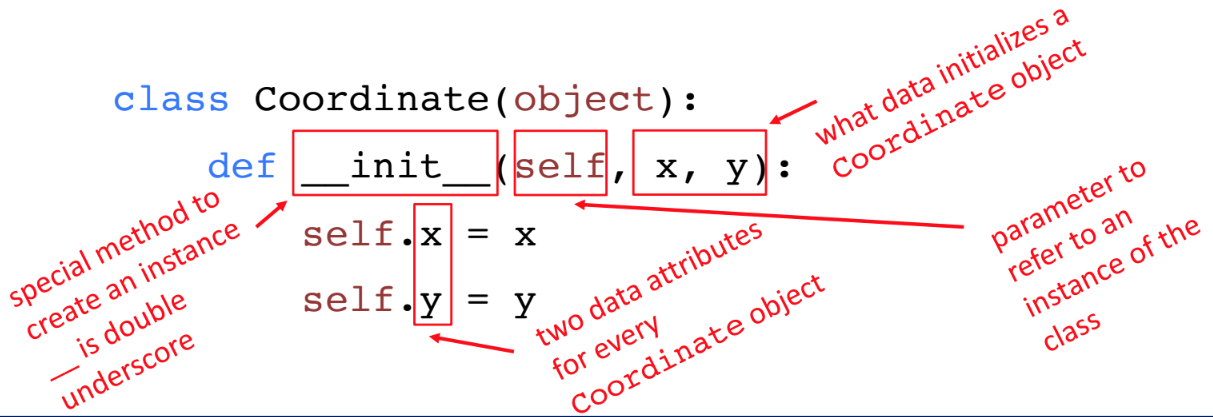
**Classes**

* a way to create your own data type using the built-in data types as building blocks
  + Real life examples: Elevator, employee, queue at store, stack of pancakes...
* attributes are data & procedures that belong to the class
  + Data attributes: objects that make up the class
  + Methods/procedural attributes: functions that only work with this class
* self:
  + Refers to the instance the method is called on
  + Always the 1st argument when defining a method
  + Not used outside the class definition
* Creating a class:

1. Define class name

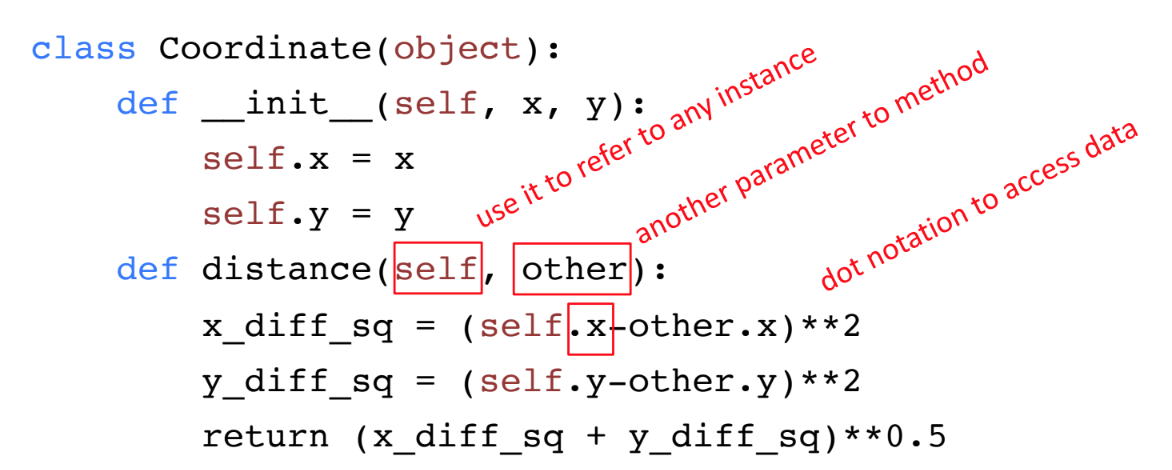


1. Define class attributes
   1. Define how to create an instance of a class using the \_\_init\_\_ method



* 1. Define other methods, these do not need to start with \_\_

These methods only work for this class.



* Using a class:
* First create a new instance of the class

# Using the example Coordinate class above

* + c1 = Coordinate(1,1)
  + c2 = Coordinate(2,1)
* Carry out operations on the instances, using class methods
  + c1.distance(c2)
* In order to be able to call print on an instance of your class you need to define the \_\_str\_\_ function.
  + there are a few more special operators ( \_\_len\_\_, \_\_eq\_\_ etc… → look at lecture slides for details on these)
* You can use isinstance() to check if an instance is an object of a class.
* In general the class defines the representation and methods common across all instances of the class; whereas an object is a SPECIFIC instance of the class
* In general you want to keep your internal representation of your class hidden, to prevent adversarial attacks and bugs.
  + the internal representation refers to what is in your \_\_init\_\_ method
* class variables: shared across all members of a class
  + defined outside of the \_\_init\_\_ method

**Getter and Setter Methods:**

● Getter and setters should be used outside of classes to access data attributes.

● It’s better style to access attributes this way, makes code easier to maintain and

helps prevent bugs

**Inheritance & Hierarchies:**

Why use inheritance?

● Allows you to extend a class with new/different capabilities.

● Reuse code

● Commonalities are explicit in the parent class, differences explicit in the subclass

Structure & how it works:

● You create an inheritance relationship between two classes by defining what

goes into the parameter in the class definition.

Graphical user interface, text, application

Description automatically generated

The child class inherits all the methods from the parent class.

● We can define new methods in the child class to extend behaviour

● We can redefine methods defined in the parent class to modify behaviour

● When you call a method on an instance of a class: the interpreter tries to find the

method at the level of a class and then checks the parent.

○ This means that if two methods with the same name are defined, the

method in the child class takes precedence.

MIT OpenCourseWare

https://ocw.mit.edu

6.100L Introduction to CS and Programming Using Python

Fall 2022

For information about citing these materials or our Terms of Use, visit: https://ocw.mit.edu/terms