* Course Overview
  + Python is an object orient programming language
* Overview
  + Naming special functions
    - \_\_feature\_\_
    - Called dunder [feature]
  + Dunder
    - Portmanteau of ‘double underscore’
  + Instance attributes
    - Defined in \_\_init\_\_
    - Assigned on per object basis
  + Ex)
    - class Rectangle:
    - def \_\_init\_\_(self, width, height)
    - self.width = width
    - self.height = height
* Class Attributes
  + Attribute that is associated with a class but not with each instance of a class
    - An attribute whose valued is shared between all instance of a class
  + Ex)
    - class MyClass:
    - my\_class\_attriute = “class attributes go here”
    - MY\_CONSTANT = “they are often class-specific contants”
    - def \_\_init\_\_(self):
    - self.my\_instance\_attribute = “instance attributes here”
  + Scopes in Python
    - Local: Inside the current function
    - Enclosing: Inside enclosing functions
    - Global: At the top level of the module
    - Built-in: In the special builtins module
  + To access class attributes you have to use ClassName.class\_attribute
  + Ex)
    - def \_\_init\_\_(self, …):
    - ….
    - ShippingContainer.next\_serial += 1
  + The Zen of Python: Explicit is better than implicit
  + Can access the class attribute outside the class with ClassName.class\_attribute
    - Can also access the class attribute through an instance
  + Ex)
    - ShippingContainer.next\_serial
    - c4.next\_serial
  + can also access class attributes inside a class using self.class\_attribute
  + ex)
    - def \_\_init\_\_(self, …):
    - ….
    - self.next\_serial += 1
  + Best to avoid self and use ClassName.class\_attribute
    - Make is clear if instance attribute and which are class attribute
  + Pitfall: although you can read class attribute through self reference, attempting to assign to a class attribute through self reference won’t have the desired effect
    - Assigning to an instance attribute is how you bring the attribute into being
    - self.class\_attribute will create an instance attribute that shadows the actual class attribute
  + There is no class scope in python
* Static Methods
  + \_method(): are implementation detail
    - Not intended for use out side
  + \_\_method\_\_(): are special methods
  + There are to two ways to associate method with the class rather than instances of the class
    - Static method decorator
  + Ex)
    - @staticmethod
    - def \_generate\_serial():
    - ….
  + Static methods are decorated with @staticmethod and they don’t require the self parameter
    - Use ClassName.\_static\_method\_name() to call static methods
    - Self can still be used to call static methods but explicit is better than implicit
  + The Static Terminology is a relic from c and c++
* Class Methods
  + Use decorate @classmethod, method accepts cls as first argument
  + Ex)
    - class MyClass:
    - attribute = “class attribute”
    - @classmethod
    - def my\_class\_method(cls, message):
    - cls.attribute = message #access class attribute via cls
  + cls plays an analogous role to self
  + @classmethod
    - Requires access to the class object to call other methods or the constructor
  + @staticmethod
    - No access needed to either class or instance objects
    - Most likely an implementation detail of the class
    - May be able to be moved outside the class to become a global-scope function in the module
  + The ‘named constructor’ idiom
    - A factory method which returns an instance of a class
    - The method name allows callers to express intent, and allows construction to be performed with different combinations of arguments
    - Originally a C++ idiom, also applicable in python
  + Ex)
    - @classmethod
    - def create\_empty(cls, owner\_code):
    - return cls(owner\_code, contents=[])
  + This allows us to support multiple constructors with different behaviors
* Static Methods with Inheritance