**Notes on “Python OOP Tutorial 1: Classes and Instances – Corey Shafer” (15:24)**

* Classes allow us to logically group our data and functions in a way that is easy to reuse and easy to build upon if need be.
* Data (aka **attributes**) and functions (aka **methods**)
* If you have a class or function that you want to leave empty, just have a pass statement (python will know to skip that)
* Class – blueprint for creating instances.
* Instance – a unique object of a class
* **Instance Variables** – Contain data that is unique to each instance
* def \_\_init\_\_(self):
  + This is just the constructor
  + When we create methods in a class, they receive the instance as the first instance automatically and you call that self. After self, you can specify other arguments yourself.
* Each method in a class takes the instance as the first argument and always call that self
* If we left off the self parameter, we get the method() takes 0 positional arguments but 1 was given

**Notes on “Python OOP Tutorial 2: Class Variables – Corey Shafer” (11:41)**

* **Class Variables** – Variables that are shared among all instances of a class
* Class variables are defined before the \_\_init\_\_ and after class classname
* When we access these class variables, we need to access them within the class itself or an instance of the class
* The namespace of an instance variable (ex: “print(emp\_1.\_\_dict\_\_)”) Prints out the attributes

**Notes on “Python OOP Tutorial 3: classmethods and staticmethods – Corey Shafer” (15:20)**

* Regular methods take in the instance as the first argument
* Class Methods have the class method decorator before the class declaration @classmethod
* Python Decorators (watch his video on it later) – Alters the functionality of our method and @classmethod receives the class (cls) as the first argument instead of the instance (self)
* You can use class methods to provide multiple ways of creating our objects. You can use class methods as alternative constructors.
* Static methods and class methods are DIFFERENT
* Static methods don’t pass in the instance or the class. They behave like regular functions. Uses decorator @staticmethod. They don’t operate on the instance or the class.
* You should use a static method if you don’t access the instance or the class anywhere within the function

**Notes on “Python OOP Tutorial 4: Inheritance – Creating Subclasses – Corey Shafer” (19:40)**