SECTION 8: OBJECT ORIENTED PROGRAMMING, 1 hour 21 minutes, 9 parts

- **4/9 Object Oriented Programming - Inheritance and Polymorphism**
  - **<u>-> inheritance and polymorphism</u>**
    - **<u>inheritance forms new classes using classes which have already been defined</u>**
  - -> the ability to reuse code

    - **-> in the .ipynb file**
    - **-> she first creates a base class**
      - def __init__(self):
        - print("Animal created") <- she's created a simple base class
      - def who_am_i(self):
        - print("I am an animal")
      - def eat(self):
        - print("I am eating")

    - **-> then she runs the code**
      - myanimal.eat()
      - myanimal.whoami()

    - **-> inheritance**
      - **class Dog(animal): <- when two classes have similar features, animal is the base class (one class is used as an input to another**
        - def __init__(self):
          - Animal.__init__(self) <- **so you can create an instance of the dog class which inherits from the base class of animal**
            - -> **the methods from the base class are inherited**
      - -> then she creates an example of the class
      - -> mydog.eat() <- **the new class based off of the base class inherits its methods**
        - **-> you can overwrite these methods**
        - -> def who_am_i(self):
          - print("I am a dog!")
          - -> creating the dog again
        - -> you can also create new methods / add them on -> **the class which inherits methods from the base class -> these methods can be overwritten or added to**
        - -> mydog.bark() e.g

    - **-> polymorphism (many classes using the same method)**
      - different object classes can share the same method name

      - **example**
        - -> she creates two new classes (a dog class and a cat class)
        - -> then defines methods (functions in the definitions of the classes) -> these print out statements

        - **-> then she creates two instances of the class**
          - -> then executes two methods
          - -> name_of_dog.speak() e.g

        - **-> these are two separate classes which use the same method**
          - **-> a cat speaks and a dog speaks**

- **-> the method is the speaking**
- **-> but we still have two separate classes which use the same method**
- **-> we can use this method they have in common when iterating through combinations of them**

- **-> using abstract classes and inheritance**
  - classes which don't expect to be instantiated
  - -> **classes which are specifically designed to be used as base classes (e.g an animal class)**
  - **-> def __init__(self, name):**
    - **self.name = name**
  - **def speak(self): <- she's defined a base class, and all of the classes which are based off of it will inherit this method (polymorphism -> the classes which are based off of this class will inherit the same method)**
    - **raise NotImplementedError("Subclass must implement this abstract method") <- this raises an error**

- **-> then she creates an instance of the class**
  - -> an error is raised
  - -> **because the class was defined to be a base class (with other classes inheriting from it) -> and for the speak method (function defined in the class) to be overwritten in the classes which inherit from it**
    - -> **each of these separate inheritances is a polymorphism**
  - -> then class Dog(Animal): <- making a class from the base class of animal
    - def speak(self):
      - return self.name+ " says woof!"
    - -> then to define an instance of the class, it's Fido = Dog("Fido") and Fido.speak

- **-> another example is to open files (open files is the base class and the different classes which are based off of that class are the names of the different file types which we want to open)**
- -> the same method name -> and the specific classes