OOP review using Python:

. **Objective of OOP.**

It is the programming paradigm where any world entity can be seen as object having characteristics and behaviours as methods and functions.

. **Take sample OOP example and base the discussion on it:**

. consider a delivery company, responsible of delivering packages and envelopes, and there are drivers and zone for each driver, so we will have several classes and methods to represent the company's project.

. **UML of the classes:**

DeliveryItem.

. several attributes.
some are private and static, other are just private
. method, constructors.

. **package and envelope** classes extends the DeliveryItem class and also have their own attributes and methods.

. **Driver class.** → represents the driver responsible of delivering the packages or the envelopes

. Objects are instances of the class where each objects will have all data fields and can behave with all the methods inside the class.

. Type of variables in class
  . In python, all variables are public by default in a class.
  . To make a variable private we use the notation as: __variable so it will be private also this applies to methods.
  . static vs non static :
    → static variables are variables shared by all the instances of a class, as it does not belong to a certain object exclusively ex: in our case the randomInteger generated is an example.
    → non static variables are the data fields of each instance of the class.

Note: . In this class DeliveryItem the variable __serialGenerator is private static
. And the __serialNumber is a private non static but set internally in the class automatically

. <u>Method types:</u>

→ **Instance methods:** are methods that operate on instances of the class, like the setters and getters. They are defined as def instanceMethod(self), called as self.methodName()

→ **Class methods:** Are methods used to belong to a class, defined as def classMethod(cls), to access these methods, use className.classMeth() These methods once called as variable = className.classMethod(), the "variable" will behave as an object of this class. In our example, the method __getNextSerial(cls) is a class method but here we defined it private since it is behaving just inside the class.

→ **Static methods:** are any other method used in the class like the helper methods or any other method not belonging to an object or a class, it is called by the className. in this example the hasInsurance(DeliveryItem) is a static method, called in the add, cancel insurance as $\underbrace{\text{DeliveryItem}}_{\text{className}}$.hasInsurance(self)

Class Structure:

Class   ClassName:

+ private state variables.

+ instance methods and constructor.

 → def __init__ (self, data fields)

 → setters: def set__ (self, datafield).

 → getters: def get__ (self) return self.data field

 → static methods

 → other instance methods

polymorphism and inheritance added later

+ class methods

. @classmethods

+ overloaded methods, like __repr__

Initialize an instance:

import className from file.py.

variable = className (attributes).

. access setters and getters if we want by variable.method, (these accessed methods are the instance methods and static methods.

. Inheritance:

. The inheritance is the concept of inheriting all aspects of a certain parent class onto a subclass and add or override functionalities.
polymorphism

. in our example we did two subclasses extending the Delivery item class, 1st class is the Package and the 2nd is the Envelope class.

→ How to extend a class?

. in the subclass, Import the file where the parent class exists and from it import the class. ex: class Envelope (Delivery Item)

. in the _init_.

↳ put all the parameters of the Parent and the child class.

↳ call the super().__init__( ) with parent class parameters

↳ call the attrs of the child class

→ How to override a method from the Parent Class

call it with the same signature of the parent one, ex: def __repr__(self), and consider the super().__repr__() can be called

- General Notes:

1. Incorrect to call super().__repr__(self) since it already knows about self. So when calling a method from super class no need to pass self to it.
2. isinstance will give true if used with DeliveryItem will give true if the item checked as Envelope or Package.
3. Incorrect to use for example:

```
def set_deliveries(self, _deliveries: list[DeliveryItem]):
    if self._deliveries is None:
```

this will give an error because here we are checking _deliveries but it's not yet initialized.

- To overcome it we use at the beginning of the method

```
if not hasattr(self, '_deliveries'):
    self._deliveries = [ ]
```

Note: Better approach in Python is to allocate all variables in the init fct, and use the setters if later on we want to change the values of the data.

- By default all is public in python. All attributes and methods are public.
- Self is not passed as an argument.
- Static method are called using parent class.method(), can be overrided if we want to change its behaviour in the child class. Also we can call it using super().method()