Question 1: Define Object Oriented Programming Language?

Answer: Object-oriented programming (OOP) is a programming language model in which programs are organized around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior. Examples of an object can range from physical entities, such as a human being that is described by properties like name and address, down to small computer programs, such as widgets. This opposes the historical approach to programming where emphasis was placed on how the logic was written rather than how to define the data within the logic.

Question 2: List down the Benefits of OOP?

Answer:

1. Modularity for easier troubleshooting

Something has gone wrong, and you have no idea where to look. Is the problem in the Widget file, or is it the WhaleFlumper? Will you have to trudge through that “sewage.c” file? Hope you commented your code!

When working with object-oriented programming languages, you know exactly where to look. “Oh, the car object broke down? The problem must be in the Car class!” You don’t have to muck through anything else.

That’s the beauty of encapsulation. Objects are self-contained, and each bit of functionality does its own thing while leaving the other bits alone. Also, this modality allows an IT team to work on multiple objects simultaneously while minimizing the chance that one person might duplicate someone else’s functionality.

2. Reuse of code through inheritance

Suppose that in addition to your Car object, one colleague needs a RaceCar object, and another needs a Limousine object. Everyone builds their objects separately but discover commonalities between them. In fact, each object is really just a different kind of Car. This is where the inheritance technique saves time: Create one generic class (Car), and then define the subclasses (RaceCar and Limousine) that are to inherit the generic class’s traits.

Of course, Limousine and RaceCar still have their unique attributes and functions. If the RaceCar object needs a method to “fireAfterBurners” and the Limousine object requires a Chauffeur, each class could implement separate functions just for itself. However, because both classes inherit key aspects from the Car class, for example the “drive” or “fillUpGas” methods, your inheriting classes can simply reuse existing code instead of writing these functions all over again.

What if you want to make a change to all Car objects, regardless of type? This is another advantage of the OO approach. Simply make a change to your Car class, and all car objects will simply inherit the new code.

3. Flexibility through polymorphism

Riffing on this example, you now need just a few drivers, or functions, like “driveCar,” driveRaceCar” and “DriveLimousine.” RaceCarDrivers share some traits with LimousineDrivers, but other things, like RaceHelmets and BeverageSponsorships, are unique.

This is where object-oriented programming’s sweet polymorphism comes into play. Because a single function can shape-shift to adapt to whichever class it’s in, you could create one function in the parent Car class called “drive” — not “driveCar” or “driveRaceCar,” but just “drive.” This one function would work with the RaceCarDriver, LimousineDriver, etc. In fact, you could even have “raceCar.drive(myRaceCarDriver)” or “limo.drive(myChauffeur).”

4. Effective problem solving

A language like C has an amazing legacy in programming history, but writing software in a top-down language is like playing Jenga while wearing mittens. The more complex it gets, the greater the chance it will collapse. Meanwhile, writing a functional-style program in a language like Haskell or ML can be a chore.

Object-oriented programming is often the most natural and pragmatic approach, once you get the hang of it. OOP languages allows you to break down your software into bite-sized problems that you then can solve — one object at a time.

This isn’t to say that OOP is the One True Way. However, the advantages of object-oriented programming are many. When you need to solve complex programming challenges and want to add code tools to your skill set, OOP is your friend — and has much greater longevity and utility than Pac-Man or parachute pants.

Question 3: Differentiate between function and method?

A **function** is a piece of code that is called by name. It can be passed data to operate on (i.e. the parameters) and can optionally return data (the return value). All data that is passed to a function is explicitly passed.

A **method** is a piece of code that is called by a name that is associated with an object. In most respects it is identical to a function except for two key differences:

1. A method is implicitly passed the object on which it was called.

2. A method is able to operate on data that is contained within the class (remembering that an object is an instance of a class - the class is the definition, the object is an instance of that data).

Question 4:

Class:

A blueprint created by a programmer for an object. This defines a set of attributes that will characterize any object that is instantiated from this class.

Object:

An instance of a class. This is the realized version of the class, where the class is manifested in the program.

Attribute:

As an object-oriented language, Python provides two scopes for attributes: class attributes and instance attributes.

Let's start with the basics:

- An instance attribute is a Python variable belonging to one, and only one, object. This variable is only accessible in the scope of this object and it is defined inside the constructor function, `__init__(self,..)` of the class.
- A class attribute is a Python variable that belongs to a class rather than a particular object. It is shared between all the objects of this class and it is defined outside the constructor function, `__init__(self,...)`, of the class.

Behavior:

Behavior is classified as the functions defined in a class to do something i.e, In class Person run() is behavior of Person.