Q1. Define the relationship between a class and its instances. Is it a one-to-one or a one-to-many partnership, for example?

One of the advantages of Object-Oriented programming language is code reuse. This reusability is possible due to the relationship b/w the classes. Object oriented programming generally support 4 types of relationships that are: inheritance , association, composition and aggregation. All these relationship is based on "is a" relationship, "has-a" relationship and "part-of" relationship.  
  
In this article we will understand all these relationships.  
  
Inheritance:  
  
Inheritance is “IS-A” type of relationship. “IS-A” relationship is a totally based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance. Inheritance is a parent-child relationship where we create a new class by using existing class code. It is just like saying that “A is type of B”. For example is “Apple is a fruit”, “Ferrari is a car”.  
  
For better understanding let us take a real world scenario.

HOD is a staff member of college.

All teachers are staff member of college.

HOD and teachers has id card to enter into college.

HOD has a staff that work according the instruction of him.

HOD has responsibility to undertake the works of teacher to cover the course in fixed time period.

Let us take first two assumptions , “HOD is a staff member of college” and “All teachers are staff member of college”. For this assumption we can create a “StaffMember” parent class and inherit this parent class in “HOD” and “Teacher” class.

class StaffMember

    {

        public StaffMember()

        {

        }

    }

    class HOD : StaffMember

    {

        public HOD()

        {

        }

    }

    class Teacher : StaffMember

    {

        public Teacher()

        {

        }

    }

Let us take an example for better understanding.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using static System.Console;

namespace Entity2

{

   class StaffMember

    {

        public int MemberId {get; set;}

        public string MemberName { get; set;}

        public string Department { get; set; }

        public StaffMember()

        {

        }

    }

    class HOD : StaffMember

    {

        public HOD()

        {

        }

        public int Course\_Completed { get; set; }

        public void Hod\_Info()

        {

            string Info = $"Member Id ={this.MemberId} \n Member Name={this.MemberName} \n Department Name={this.Department} \n Total Course Completed ={this.Course\_Completed} %";

            WriteLine(Info);

        }

    }

    class Teacher : StaffMember

    {

        public Teacher()

        {

        }

        public int Hod\_Id { get; set; }

        public void Teacher\_Info()

        {

            string Info = $"Member Id ={this.MemberId} \n Member Name={this.MemberName} \n Department Name={this.Department} \n Id of HOD ={this.Hod\_Id} ";

            WriteLine(Info);

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            HOD Obj\_Hod = new HOD();

            Obj\_Hod.MemberId = 10;

            Obj\_Hod.MemberName = "Dazy Arya";

            Obj\_Hod.Department = "CSE";

            Obj\_Hod.Course\_Completed = 85;

            Teacher Obj\_Tech = new Teacher();

            Obj\_Tech.Department = "CSE";

            Obj\_Tech.MemberId = 15;

            Obj\_Tech.MemberName = "Ambika Gupta";

            Obj\_Tech.Hod\_Id = 10;

            Obj\_Hod.Hod\_Info();

            Obj\_Tech.Teacher\_Info();

            ReadLine();

        }

    }

}

Q2. What kind of data is held only in an instance?

Instance data is defined in the data division in the object paragraph of a class definition and is processed by procedural code in the instance methods of that class. Instance data is organized into logical records and independent data description entries in the same manner as program data.

In [class-based](https://en.wikipedia.org/wiki/Class-based), [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), an instance variable is a [variable](https://en.wikipedia.org/wiki/Variable_(programming)) defined in a [class](https://en.wikipedia.org/wiki/Class_(computer_programming)) (i.e. a [member variable](https://en.wikipedia.org/wiki/Member_variable)), for which each instantiated [object](https://en.wikipedia.org/wiki/Object_(computer_science)) of the class has a separate copy, or [instance](https://en.wikipedia.org/wiki/Instance_(computer_science)). An instance variable has similarities with a [class variable](https://en.wikipedia.org/wiki/Class_variable),[[1]](https://en.wikipedia.org/wiki/Instance_variable#cite_note-1) but is non-[static](https://en.wikipedia.org/wiki/Static_variable). An instance variable is a variable which is declared in a class but outside of [constructors](https://en.wikipedia.org/wiki/Constructor_(object-oriented_programming)), [methods](https://en.wikipedia.org/wiki/Method_(computer_programming)), or [blocks](https://en.wikipedia.org/wiki/Block_(programming)). Instance variables are created when an object is instantiated, and are accessible to all the constructors, methods, or blocks in the class. [Access modifiers](https://en.wikipedia.org/wiki/Access_modifiers) can be given to the instance variable.

An instance variable is not a [class variable](https://en.wikipedia.org/wiki/Class_variable) although there are similarities. It is a type of [class attribute](https://en.wikipedia.org/wiki/Class_(computer_science)#Structure) (or class property, [field](https://en.wikipedia.org/wiki/Field_(computer_science)), or data member). The same dichotomy between instance and class members applies to methods ("member functions") as well; a class may have both [instance methods](https://en.wikipedia.org/wiki/Instance_method) and [class methods](https://en.wikipedia.org/wiki/Class_method).

Each instance variable lives in memory for the [lifetime](https://en.wikipedia.org/wiki/Object_lifetime) of the object it is owned by.[[2]](https://en.wikipedia.org/wiki/Instance_variable#cite_note-2)

Variables are properties an object knows about itself. All instances of an object have their own copies of instance variables, even if the value is the same from one object to another. One object instance can change values of its instance variables without affecting all other instances. Instance variables can be used by all methods of a class unless the method is declared as static.[[3]](https://en.wikipedia.org/wiki/Instance_variable#cite_note-3)

Example[[edit](https://en.wikipedia.org/w/index.php?title=Instance_variable&action=edit&section=1)]

struct Request {

static int count1; // variable name is not important

int number;

Request() {

number = count1; // modifies the instance variable "this->number"

++count1; // modifies the class variable "Request::count1"

}

};

int Request::count1 = 0;

In this C++ example, the instance variable Request::number is a copy of the class variable Request::count1 where each instance constructed is assigned a sequential value of count1 before it is [incremented](https://en.wikipedia.org/wiki/Increment_operator). Since number is an instance variable, each Request object contains its own distinct value; in contrast, there is only one object Request::count1 available to all instances with the same value.

Q3. What kind of knowledge is stored in a class?

Knowledge is the key to the success of an individual and the organization. Here are the 4 types of knowledge

In every organizational process, knowledge is essential, whether hiring staff or training the employees. Therefore, it is more crucial for the organization to build a knowledge management plan.

But before that, we must understand the types of knowledge. These include: explicit (documented information), implicit (applied information), tacit (understood information) and factual (based upon facts).

All these different types of knowledge work together to create the spectrum of how we pass information to each other and later learn and grow. So first, it’s important to note that it’s needed to understand the types of knowledge in the organization.

Design the knowledge management solution for long-term success by learning about the three main types of knowledge and how to acquire each in the organization.

When we make a knowledge management strategy, one should consider the differences between all the types of knowledge, and a practical base should be formed, which is helpful for both the short and long terms.

Explicit Knowledge

It is the most basic form of knowledge and is easy to pass along as it’s written down and very accessible. Explicit knowledge is called when data is processed, organized, structured, and interpreted. Explicit knowledge can be easily articulated, recorded, communicated, and stored, most notably in knowledge management.

For example, open a knowledge management platform and take a look around. The company data sheets, white papers, research reports, etc., are all explicit company knowledge.

Explicit knowledge consists of information like:

SOPs

Instruction manuals

Guides

White papers

Datasheets

Product specifications

Case studies

Formal documentation

For example, it’s a universal truth that there are 26 letters in the alphabet. Therefore, it is a true statement of explicit knowledge. Likely, Force = mass\*acceleration is a true statement that shows our understanding of the universe, as it is a fundamental law of physics.

Implicit Knowledge

Implicit knowledge is the practical application of explicit knowledge. There are likely instances of implicit knowledge all around the organization.

For example, let’s say, consider asking a team member how to perform a task. This could ignite a conversation about the range of options to perform the task and potential outcomes, leading to a thoughtful process to get the best course of action. The team member’s implicit knowledge educates the conversation of how to do something and what can happen further.

Also, excellent best practices and transferable skills from job to job are prime examples of implicit knowledge.

Implicit knowledge includes data gleaned from:

Skype

Email

Intranet

Meeting notes

For example, implicit knowledge can be the location of the closest mall or superstore to a house. Think that a new neighbor moves in and asks where they can get groceries. Later you share implicit knowledge when you tell them about the grocery store two streets over.

Tacit Knowledge

Tacit knowledge is the knowledge that we procure from personal experience and context. It’s information that, if any time, would be very difficult to note down, articulate, or show in a tangible form.

For example, think of learning how to make your mother’s famous recipes. Indeed, she gave you the recipe book, but when you execute it on your own, you feel something is missing. After many years of experience, she has learned the right feel for the dough, or precisely how long something should be in the oven. It’s not the points she can write down; she can feel it.

In workplaces, tacit knowledge is the application of implicit knowledge that’s completely specific to your company. As employees move from one job to another, the application of their implicit knowledge will change depending upon what’s unique about business. For example, a sales representative now knows the skill of giving a great demo and knows the specific buying signs while talking to prospects.

It’s entirely possible for tacit document knowledge, but it can be challenging. To make use of your team’s tacit knowledge, invest in strategies like:

Mentoring

Exit interviews

Ongoing quarterly interviews

This knowledge is seldom seen in generations who have consistently grown up around computers and seem to know-how gadgets, and gizmos work in their way intuitively.

Factual Knowledge

This form of knowledge is based upon facts and can be learned after a good amount of exposure. You cannot store everything in your memory, and so factual knowledge is available primarily in

Books

Online portals

Journals

A person with all the facts about the topic

For achieving your goals, you must be aware of the facts.

For instance, while giving the presentation, a business person should know the details about the facts and figures such as sales performance, current market scenario, budget, and more.

Q4. What exactly is a method, and how is it different from a regular function?

Functions can be called only by its name, as it is defined independently. But methods can't be called by its name only, we need to invoke the class by a reference of that class in which it is defined, i.e. method is defined within a class and hence they are dependent on that class.

A function is a set of instructions or procedures to perform a specific task, and a method is a set of instructions that are associated with an object.

A function is a set of instructions or procedures to perform a specific task, and a method is a set of instructions that are associated with an object.

A function is used to split the code into easily understandable parts, which can be reused as well.

Differences

Some differences between a function and method are listed below:

A function doesn’t need any object and is independent, while the method is a function, which is linked with any object.

We can directly call the function with its name, while the method is called by the object’s name.

Function is used to pass or return the data, while the method operates the data in a class.

Function is an independent functionality, while the method lies under object-oriented programming.

In functions, we don’t need to declare the class, while to use methods we need to declare the class.

Functions can only work with the provided data, while methods can access all the data provided in the given class.

Q5. Is inheritance supported in Python, and if so, what is the syntax?

Multi-Level inheritance is possible in python like other object-oriented languages. Multi-level inheritance is archived when a derived class inherits another derived class. There is no limit on the number of levels up to which, the multi-level inheritance is archived in python.

Syntax

class class1:

    <class-suite>

class class2(class1):

    <class suite>

class class3(class2):

    <class suite>

Q6. How much encapsulation (making instance or class variables private) does Python support?

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object’s variable can only be changed by an object’s method. Those types of variables are known as private variables.

A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc

Encapsulation is one of the fundamental concepts in [object-oriented programming](https://pynative.com/python/object-oriented-programming/) (OOP), including abstraction, inheritance, and polymorphism. This lesson will cover what encapsulation is and how to implement it in Python.

After reading this article, you will learn:

Encapsulation in Python

Need for Encapsulation

Data Hiding using public, protected, and private members

Data Hiding vs. Encapsulation

Getter and Setter Methods

Benefits of Encapsulation

Q7. How do you distinguish between a class variable and an instance variable?

Following are the notable differences between Class (static) and instance variables.

|  |  |
| --- | --- |
| Instance variables | Static (class) variables |
| Instance variables are declared in a class, but outside a method, constructor or any block. | Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block. |
| Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed. | Static variables are created when the program starts and destroyed when the program stops. |
| Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. ObjectReference.VariableName. | Static variables can be accessed by calling with the class name ClassName.VariableName. |
| Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class. | There would only be one copy of each class variable per class, regardless of how many objects are created from it. |

Q8. When, if ever, can self be included in a class's method definitions?

If you are working with [Python](https://www.edureka.co/blog/python-tutorial/), there is no escaping from the word “self”. It is used in method definitions and in variable initialization. The self method is explicitly used every time we define a method. In this article, we will get into the depth of self in Python in the following sequence:

[What is the use of self in Python?](https://www.edureka.co/blog/self-in-python/#what)

[Python Class self Constructor](https://www.edureka.co/blog/self-in-python/#self)

[Is self in Python a Keyword?](https://www.edureka.co/blog/self-in-python/#keyword)

What is the use of Self in Python?

The self is used to represent the [instance](https://www.edureka.co/blog/isinstance-in-python/) of the class. With this keyword, you can access the attributes and methods of the [class in python](https://www.edureka.co/blog/python-class/). It binds the attributes with the given arguments. The reason why we use self is that Python does not use the ‘@’ syntax to refer to instance attributes. Join our [Master Python programming](https://www.edureka.co/masters-program/python-developer-training) course to know more. In Python, we have methods that make the instance to be passed automatically, but not received automatically.

Q9. What is the difference between the \_ \_add\_ \_ and the \_ \_radd\_ \_ methods?

There is no difference. One performs a+b , the other b+a . Likewise rmul and mul produce the same result, but rsub and sub & rdiv and div do not.

Q10. When is it necessary to use a reflection method? When do you not need it, even though you support the operation in question?

Java Reflection provides ability to inspect and modify the runtime behavior of application. Reflection in Java is one of the advance topic of core java. Using java reflection we can inspect a class, [interface](https://www.digitalocean.com/community/tutorials/interface-in-java), [enum](https://www.digitalocean.com/community/tutorials/java-enum" \o "Java Enum), get their structure, methods and fields information at runtime even though class is not accessible at compile time. We can also use reflection to instantiate an object, invoke it’s methods, change field values.

Reflection in Java

Reflection in Java is a very powerful concept and it’s of little use in normal programming but it’s the backbone for most of the Java, J2EE frameworks. Some of the frameworks that use java reflection are:

JUnit - uses reflection to parse [@Test](https://www.digitalocean.com/community/users/test) annotation to get the test methods and then invoke it.

Spring - dependency injection, read more at [Spring Dependency Injection](https://www.digitalocean.com/community/tutorials/spring-dependency-injection)

Tomcat web container to forward the request to correct module by parsing their web.xml files and request URI.

Eclipse auto completion of method names

[Struts](https://www.digitalocean.com/community/tutorials/struts-2)

Hibernate

The list is endless and they all use java reflection because all these frameworks have no knowledge and access of user defined classes, interfaces, their methods etc. We should not use reflection in normal programming where we already have access to the classes and interfaces because of following drawbacks.

Poor Performance - Since java reflection resolve the types dynamically, it involves processing like scanning the classpath to find the class to load, causing slow performance.

Security Restrictions - Reflection requires runtime permissions that might not be available for system running under security manager. This can cause you application to fail at runtime because of security manager.

Security Issues - Using reflection we can access part of code that we are not supposed to access, for example we can access private fields of a class and change it’s value. This can be a serious security threat and cause your application to behave abnormally.

High Maintenance - Reflection code is hard to understand and debug, also any issues with the code can’t be found at compile time because the classes might not be available, making it less flexible and hard to maintain.

Java Reflection for Classes

In java, every object is either a primitive type or reference. All the classes, enums, arrays are reference types and inherit from java.lang.Object. Primitive types are - boolean, byte, short, int, long, char, float, and double. java.lang.Class is the entry point for all the reflection operations. For every type of object, [JVM](https://www.digitalocean.com/community/tutorials/difference-jdk-vs-jre-vs-jvm) instantiates an [immutable](https://www.digitalocean.com/community/tutorials/how-to-create-immutable-class-in-java) instance of java.lang.Class that provides methods to examine the runtime properties of the object and create new objects, invoke its method and get/set object fields. In this section, we will look into important methods of Class, for convenience, I am creating some classes and interfaces with [inheritance](https://www.digitalocean.com/community/tutorials/inheritance-java-example) hierarchy.

Q11. What is the \_ \_iadd\_ \_ method called?

The Python \_\_iadd\_\_() magic method implements in-place addition x += y that adds together the operands and assigns the result to the left operand. This operation is also called augmented arithmetic assignment. The method simply returns the new value to be assigned to the first operand

The Python \_\_iadd\_\_() magic method implements [in-place addition](https://blog.finxter.com/python-in-place-addition-operator/) x += y that adds together the operands and assigns the result to the left operand. This operation is also called augmented arithmetic assignment. The method simply returns the new value to be assigned to the first operand.

When you call x += y, Python first attempts to call x.\_\_iadd\_\_(y).

If this is not implemented, it tries the normal addition [x.\_\_add\_\_(y)](https://blog.finxter.com/python-__add__/" \o "Python __add__() Magic Method" \t "_blank).

If this is not implemented either, it tries reverse addition [y.\_\_radd\_\_(x)](https://blog.finxter.com/python-__rxadd__-magic-method/) with swapped operands.

The result is then assigned to the first operand x. If none of those operations is implemented, Python raises a TypeError.

We call this a [“Dunder Method”](https://blog.finxter.com/python-list-of-dunder-methods/) for “Double Underscore Method” (also called “magic method”). To get a [list of all dunder methods](https://blog.finxter.com/python-list-of-dunder-methods/) with explanation, check out our [dunder cheat sheet article](https://blog.finxter.com/python-dunder-methods-cheat-sheet/" \o "Python Dunder Methods Cheat Sheet" \t "_blank) on this blog.

Basic Example Overriding \_\_iadd\_\_

In the following code example, you create a class Data and define the magic method \_\_iadd\_\_(self, other).

Q12. Is the \_ \_init\_ \_ method inherited by subclasses? What do you do if you need to customize its behavior within a subclass?

The \_\_init\_\_ method is the Python equivalent of the C++ constructor in an object-oriented approach. The \_\_init\_\_ function is called every time an object is created from a class. The \_\_init\_\_ method lets the class initialize the object's attributes and serves no other purpose. It is only used within classes.

How Does the \_\_init\_\_ Method Work?

The \_\_init\_\_ method is the Python equivalent of the [C++ constructor](https://www.udacity.com/blog/2021/03/what-is-a-constructor-in-c.html?utm_source=rss&utm_medium=rss&utm_campaign=what-is-a-constructor-in-c) in an object-oriented approach. The \_\_init\_\_  function is called every time an object is created from a class. The \_\_init\_\_ method lets the class initialize the object’s attributes and serves no other purpose. It is only used within classes.

Create a Class

Let’s begin by creating a class:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | class Dog:    def \_\_init\_\_(self,dogBreed,dogEyeColor):        self.breed = dogBreed      self.eyeColor = dogEyeColor... |

First, we declare the class Dog using the keyword class. We use the keyword def to define a function or method, such as the \_\_init\_\_ method. As you can see, the \_\_init\_\_ method initializes two attributes: breed and eyeColor.

We’ll now see how to pass these parameters when declaring an object. This is where we need the keyword self to bind the object’s attributes to the arguments received.

Create an Object

Next we’ll create an object, or instance, of the class Dog:

|  |  |
| --- | --- |
| 1 | ...Tomita = Dog("Fox Terrier","brown")... |

When we create the object tomita (which is the dog’s name), we first define the class from which it is created (Dog). We next pass the arguments “Fox Terrier” and “brown,” which correspond to the respective parameters of the \_\_init\_\_ method of the class Dog.

The \_\_init\_\_ method uses the keyword self to assign the values passed as arguments to the object attributes self.breed and self.eyeColor.

Access Object Attributes

To access an attribute of your brand new Fox Terrier object, you can use the dot (.) notation to get the value you need. A print statement helps us demonstrate how this works:

|  |  |
| --- | --- |
| 1 | ...print("This dog is a",tomita.breed,"and its eyes are",tomita.eyeColor) |

Executing the above code gives us the following result:

|  |  |
| --- | --- |
| 1 | This dog is a Fox Terrier and its eyes are brown |

The program accessed tomita’s attributes and displayed them properly.

The Default \_\_init\_\_ Constructor

In Python, a constructor does not necessarily need parameters passed to it. There can be default parameters. A constructor with no mandatory parameters is called a default constructor. Let’s rewrite our class with a default constructor:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | class Dog:    def \_\_init\_\_(self, dogBreed="German Shepherd",dogEyeColor="Brown"):        self.breed = dogBreed      self.eyeColor = dogEyeColor |

If a user does not enter any values, the constructor will assign “German Shepherd” and “Brown” as the attributes.

We can now create an instance of Dog without specifying any parameter:

|  |  |
| --- | --- |
| 1 | tomita = Dog() |

Since there are no arguments to pass, we use empty parentheses after the class name. We can still display the object’s attributes:

|  |  |
| --- | --- |
| 1 | print("This dog is a",tomita.breed,"and its eyes are",tomita.eyeColor) |

This gives us the following output:

|  |  |
| --- | --- |
| 1 | This dog is a German Shepherd and its eyes are Brown |

The “[self](https://blog.finxter.com/self-in-python/)” argument is the default argument of each method and it refers to the object on which it is called—in our case, the first operand of the in-place operation.

The “other” argument of the in-place method refers to the second operand, i.e., y in the in-place operation x += y.

The return value of the operation returns a dummy string 'finxter 42' to be assigned to the first operand. In practice, this would be the result of the in-place addition.