Q1. What Is Object Oriented Programming?

Ans. As the name suggests, Object-Oriented Programming or OOPs refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

**OOPs Concepts:**

* Class
* Objects
* Data Abstraction
* Encapsulation
* Inheritance
* Polymorphism
* Dynamic Binding
* Message Passing

Q2. What Are Properties Of Object Oriented Systems?

Ans. [**Properties of Object Oriented Systems include**](https://www.bing.com/ck/a?!&&p=b176dce612a0f0adJmltdHM9MTY5NjIwNDgwMCZpZ3VpZD0xM2VmNjExMS1mOGNmLTZlMzctMTEwYS03MjQ1Zjk3ZDZmZDUmaW5zaWQ9NTcyMA&ptn=3&hsh=3&fclid=13ef6111-f8cf-6e37-110a-7245f97d6fd5&psq=What+Are+Properties+Of+Object+Oriented+Systems%3f+&u=a1aHR0cHM6Ly93d3cuZWR1Y2F0aXZlLmlvL2Jsb2cvb2JqZWN0LW9yaWVudGVkLXByb2dyYW1taW5n&ntb=1):

* Reproducible, simple structures
* Reusability of objects across programs
* Polymorphism, allowing for class-specific behavior
* Easier debugging, as classes often contain all applicable information to them
* Secure protection of sensitive information through encapsulation
* Abstraction
* Encapsulation
* Modularity
* Objects have state and behaviors, with properties representing their state and methods modifying or operating on these properties.

Q3. What Is Difference Between Class And Interface?

Ans. In PHP, **classes** and **interfaces** are both used to define objects and their behavior. However, there are some differences between them.

A **class** is a blueprint for creating objects. It defines the properties and methods that an object of that class will have. Properties are variables that hold data, while methods are functions that perform actions on the object.

An **interface**, on the other hand, is a collection of abstract methods. It defines the methods that a class must implement if it implements that interface. An interface does not define any properties.

One key difference between classes and interfaces is that a class can implement multiple interfaces, but it can only inherit from one class. This is known as **multiple inheritance**.

Another difference is that a class can have both abstract and non-abstract methods, while an interface can only have abstract methods.

In summary, classes define objects and their behavior, while interfaces define the methods that a class must implement if it implements that interface.

Q4. What Is Overloading?

Ans. In PHP, overloading is the ability to create multiple functions or properties with the same name but different implementations. Function overloading is not supported in PHP, but method overloading can be achieved using the magic method \_\_call() 123.

The \_\_call() method is invoked when an inaccessible or non-existent method is called on an object. It takes two parameters: the name of the method being called and an array of arguments passed to the method. Using this method, you can create a single method that can handle multiple function calls with different parameters.

Q5. What Is T\_PAAMAYIM\_NEKUDOTAYIM (Scope Resolution Operator (::) with Example ?

Ans. In PHP, T\_PAAMAYIM\_NEKUDOTAYIM is the name of the scope resolution operator (::). It is a token that allows access to static, constant, and overridden properties or methods of a class

The name T\_PAAMAYIM\_NEKUDOTAYIM is Hebrew for “double colon” . It was chosen by the Zend team while writing the Zend Engine 0.5, which powers PHP.

Q6. What are the differences between abstract classes and interfaces?

Ans. Abstract classes and interfaces are both used in object-oriented programming to define blueprints for classes, but they have distinct differences in terms of their structure and usage.

1. **Definition and Structure:**
   * An abstract class is a class that may have some implemented methods alongside abstract (unimplemented) methods. It may also have instance variables and constructors.
   * An interface is a blueprint of a class that only includes method signatures (no method implementations) and constant fields (constants).
2. **Method Implementation:**
   * An abstract class can have both abstract and concrete (implemented) methods.
   * An interface can only have method signatures (method names, parameters, return types) with no method implementations.
3. **Inheritance:**
   * A class can extend only one abstract class (single inheritance).
   * A class can implement multiple interfaces (multiple inheritance in a limited sense).
4. **Usage:**
   * Abstract classes are used when you want to provide a common implementation for a group of related classes and have some methods that need to be implemented by derived classes.
   * Interfaces are used to define contracts that multiple unrelated classes can adhere to, ensuring they provide specific methods, but leaving the implementation to the individual classes.
5. **Fields:**
   * Abstract classes can have instance variables, constructors, and any other members that a regular class can have.
   * Interfaces can only have constant fields (static final) and no instance variables.
6. **Extending or Implementing:**
   * To use an abstract class, a class must extend it using the **extends** keyword.
   * To use an interface, a class must implement it using the **implements** keyword.
7. **Default Implementations:**
   * Java 8 and later versions allow interfaces to have default method implementations, providing a level of method implementation in interfaces. Abstract classes have no concept of default methods.

In summary, abstract classes are used to provide a common structure and some implementation details to related classes, while interfaces are used to define contracts that unrelated classes can adhere to, ensuring they have certain methods without providing any implementation details.

Q7. Define Constructor and Destructor?

Ans. In object-oriented programming, constructors and destructors are special methods associated with classes and are used for object initialization and object cleanup, respectively.

1. **Constructor:**
   * A constructor is a special method within a class that is automatically called when an instance (object) of the class is created.
   * Its primary purpose is to initialize the state (data members) of an object and set up any necessary resources.
   * Constructors have the same name as the class and do not have a return type, not even **void**.
   * Constructors can be overloaded, allowing a class to have multiple constructors with different parameter lists.
2. Destructor:

A destructor is a special method (in some programming languages) that is automatically called when an object is no longer needed or goes out of scope.

The purpose of a destructor is to perform cleanup actions, release resources, and deallocate memory associated with the object.

Some programming languages, like C++, have explicit destructors, while others, like Java, do not have destructors in the same sense due to automatic memory management (garbage collection).

In languages with automatic garbage collection, the cleanup is handled automatically, and there's no need for an explicit destructor.

Q 8. How to Load Classes in PHP?

Ans. In PHP, classes are typically loaded using a combination of autoloading and manual inclusion of files. Autoloading is a mechanism that automatically includes the necessary PHP files when a class is first used or instantiated. This approach helps manage the loading of classes more efficiently and reduces the need for manual **require** or **include** statements for each class.

Here's a step-by-step approach to loading classes in PHP:

1. **Autoloading Classes:** PHP provides a built-in function called **spl\_autoload\_register** that allows you to register a custom autoloader function. This function is invoked whenever a class is accessed or instantiated but has not yet been loaded.
2. **Class File Naming and Structure:** Follow a naming convention for your class files that matches the class name. For example, if you have a class named **MyClass**, the file should be named **MyClass.class.php**.
3. **Use the Class:** When you use or instantiate a class, PHP will automatically call the registered autoloader function, which will include the necessary class file.

PHP will automatically load the **MyClass** class by calling the autoloader function (**my\_autoloader** in this example) and including the appropriate file.

1. **PSR-4 Autoloading (Optional):** PSR-4 is a standard for autoloading classes, widely used in PHP projects. It specifies a namespace-to-directory mapping that allows for a more organized class loading approach.

You can use third-party libraries such as Composer, which provides a robust autoloading mechanism based on PSR-4 standards.

The autoloaded classes will be loaded automatically based on the PSR-4 mapping.

By using autoloading mechanisms like **spl\_autoload\_register** or utilizing PSR-4 autoloading, you can efficiently load classes in PHP without the need for manual inclusion of class files.

Q 9. How to Call Parent Constructor?

Ans. In object-oriented programming, calling the parent class's constructor from a subclass constructor is essential when you want to include the initialization logic defined in the parent class's constructor in addition to the subclass-specific initialization. This process ensures that both the parent and subclass's initialization logic is executed properly.

Here's how to call the parent class's constructor from a subclass constructor in PHP:

1. **Using parent::\_\_construct() in Subclass Constructor:**

In PHP, you can use the **parent::\_\_construct()** statement within the subclass constructor to call the constructor of the parent class.

Q10. Are Parent Constructor Called Implicitly When Create An ObjectOf Class?

Ans. When a child class does not define a constructor, the parent constructor is called implicitly.

However, if the child class defines a constructor, the parent constructor is not called implicitly. In order to run a parent constructor, a call to **parent::\_\_construct()** within the child constructor is required

If the derived class has any input parameters in common with the parent class, then it should invoke the **super()** function on those parameters so they get sent to the parent class for instantiation

If the parent class implements a constructor with arguments and has no constructor with no arguments, then the child constructors must explicitly call a parent's constructor.

In summary, whether the parent constructor is called implicitly when creating an object of a child class depends on whether the child class defines a constructor. If it does, the parent constructor must be called explicitly within the child constructor.

Q11. What Happen, If Constructor Is Defined As Private Or Protected?

Ans. If a constructor is defined as private, it can only be accessed within the class and cannot be called from outside the class. This means that objects of the class cannot be created from outside the class. However, the class can still have static methods that can create objects of the class.

If a constructor is defined as protected, it can only be accessed within the class and its derived classes. This means that objects of the class cannot be created from outside the class or its derived classes. However, derived classes can still have constructors that call the base class’s protected constructor

Q12. What are PHP Magic Methods/Functions? List them Write program for Static Keyword in PHP?

Ans. PHP Magic Methods are special methods that override PHP’s default action when certain actions are performed on an object. The following method names are considered magical: \_\_construct(), \_\_destruct(), \_\_call(), \_\_callStatic(), \_\_get(), \_\_set(), \_\_isset(), \_\_unset(), \_\_sleep(), \_\_wakeup(), \_\_serialize(), \_\_unserialize(), \_\_toString(), \_\_invoke(), \_\_set\_state(), \_\_clone(), and \_\_debugInfo()