**PYTHON OOPS ASSIGNMENT**

**Q1.** What is the purpose of Python's OOP?

**Ans.**  Object-oriented Programming (OOPs) in Python, is a programming paradigm that uses objects and classes rather than function or logic in programming.  Object is a unit or block that comes with some unique characteristics (data) and methods. The object's data are its properties, that defines what it is, and the object's methods, are its functions, that defines what the object can do. It is also, an approach used for creating neat and reusable code instead of a redundant one. The program is divided into self-contained objects or several mini-programs. Every Individual object represents a different part of the application having its own logic and data to communicate within themselves.

**Q2.** Where does an inheritance search look for an attribute?

**Ans.**

**Q3**. How do you distinguish between a class object and an instance object?

**Ans.** There are two types of objects in python.

* Class object
* Instance object

|  |  |
| --- | --- |
| **Class object** | **Instance object** |
| * Class is used as a template for declaring and creating the objects. * A class is like a blueprint. * When we **create**a class in python then a **class object** is created. * Whenever python encounters a class statement in the whole program then it creates a class object and assigns a name to that object i.e., class name. * Class object inherits the attributes of its parent classes. * When a class is created, no memory is allocated. | * When we **call**a class, it creates an instance object of that class from which the object has been created. * Instance is a copy of the class with actual values. * Instance objects are real objects in your python code. * The instance object has access to attributes of the class from which it is created. * Instance object inherits the attributes of the class object from which it was created. * Objects are allocated memory space whenever they are created. |

**Q4.** What makes the first argument in a class’s method function special?

**Ans.** The first argument of every class method, including init is always named as self. It is the first parameter to the Instance method and constructor. If you don’t provide it, it will cause an error.

Self represents the instance of the class. By using the “self” we can access the attributes and methods of the class in python. It binds the attributes with the given arguments. Self is always pointing to Current Object. It is not a keyword and the user can use a different parameter name in place of it.

**Q5.** What is the purpose of the **init** method?

**Ans.** *"\_\_init\_\_"* is a reserved method in python classes. The *\_\_init\_\_()* doesn’t create an object but is automatically called after the object is created. Hence, the *\_\_init\_\_()* is not a constructor. Since Python will automatically call the *\_\_init\_\_()* method immediately after creating a new object, you can use the *\_\_init\_\_()* method to initialize the object’s attributes. If the *\_\_init\_\_* has parameters other than the self, you need to pass the corresponding arguments when creating a new object.

**Q6.** What is the process for creating a class instance?

**Ans.** An instance of a class is an object. A class needs to be instantiated if we want to use the class attributes in another class or method. To create instances of a class, you call the class using class name and pass in whatever arguments its *\_\_init\_\_* method accepts.  All the instances share the attributes and the behavior of the class.

The syntax to create the instance of the class is given below:

*Object\_name = class\_name (arguments)*

**Q7.** What is the process for creating a class?

**Ans.** In Python, a class can be created by using the keyword *class*, followed by the *class name*. The syntax to create a class is given below:

*class ClassName:*

*# Statement*

**Q8.** How would you define the super classes of a class?

**Ans.** The class whose properties gets inherited by another class is known as ***superclass or parent class***. the class which inherits the properties of another class is known as the ***subclass or child class***. A subclass inherits all data and behavior of parent class.

***Inheritance*** is the property of an OOP language through which the data and behavior of a superclass can be passed onto a subclass. Inheritance is one of the mechanisms to achieve the re-use property of oops.

**Q9.** What is the relationship between classes and modules?

**Ans.** Python code is organized in files of *.py* extension called "modules" and groups of related modules called “packages". A module is a distinct unit that may have one or more closely-related classes, functions and variables. Modules need to be imported before they are read and used. To access the functions inside the module the dot(.) operator is used.

A class is used to define a blueprint for a given object, whereas a module is used to reuse a given piece of code inside another program. There is no limit on how many classes one can put in a module.

**Q10.** How do you make instances and classes?

**Ans.**

**Q11.** Where and how should be class attributes created?

**Ans. Class attributes** are the variables defined directly in the class that are shared by all objects of the class. Such attributes are defined in the class body parts usually at the top, for legibility. To define a class attribute, you place it outside of the \_\_init\_\_() method. One can access the class attribute via instances of the class or via the class name.

# Class attribute  
class Student:  
 count = 0  
 def \_\_init\_\_(self):  
 Student.count += 1

The ***count*** variable acts as a class attribute. **Class attributes** remain the same for every object.

**Q12.** Where and how are instance attributes created?

**Ans. Instance attributes** are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor using the self parameter i.e., they are defined in the \_\_init\_\_() function. Instance attributes are not shared by objects. Every object has its own copy of the instance attribute.

# Class attribute & instance attribute  
class Student:  
 count = 0  
  
 def \_\_init\_\_(self, name, course):  
 self.name = name  
 self.course = course  
 Student.count += 1  
  
  
Student1 = Student("manu", "Big Data")  
Student2 = Student("Diya", "Python")  
  
print(Student1.name)  
print(Student2.name)

print(Student1.count)  
print(Student2.count)

Output

manu

Diya

2

2

Here name and course are instance attributes.

**Q13.** What does the term "self" in a Python class mean?

**Ans.**  Self represents the instance of the class. By using the “self” we can access the attributes and methods of the class in python. It binds the attributes with the given arguments. Self is always pointing to Current Object. It is not a keyword and the user can use a different parameter name in place of it.

**Q14.** How does a Python class handle operator overloading?

**Ans.** Operator overloading is the process of using an operator in different ways depending on the operands. One can change the way an operator in Python works on different data-types.

The user can overload all the existing operators, but they cannot create any new operator. Python provides some special functions, called magic functions for performing operator overloading. Magic methods in Python are special methods that begin and end with a ***double underscore( \_\_ ).***

When we use an operator on user-defined data types then automatically a magic function associated with that operator is invoked. Changing the behavior of operator is as simple as changing the behavior of a method or function. You define methods in your class and operators work according to that behavior defined in methods.

**Q15.** When do you consider allowing operator overloading of your classes?

**Ans.** Consider two objects of a user-defined data type (physical representation of class), now if we try to add two objects with binary ‘+’ operator the compiler will throw an error because the compiler doesn’t know how to add two objects. To resolve this problem, we can define a method to overload the + operator to change its functionality as per our needs.

To overload the ‘+' operator, we will use the *\_\_add\_\_* magic method.

**Q16.** What is the most popular form of operator overloading?

### **Ans.**

1. Binary operators overloading
2. Comparison operators overloading
3. Assignment operators overloading
4. Unary operators overloading

**Q17.** What are the two most important concepts to grasp in order to comprehend Python OOP code?

**Ans.** Two key concepts of OOPs are inheritance and polymorphism.

Inheritance is one of the mechanisms to achieve the property of reusability. Inheritance allows programmer to create a general or a base class first and then later extend it to more specialized class.  The class that derives properties is called the derived class or child class and the class from which the properties are being derived is called the base class or parent class. It is transitive in nature, i.e., if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

*Class BaseClass:*

*{Body}*

*Class DerivedClass(BaseClass):*

*{Body}*

**Q18.** Describe three applications for exception processing.

**Ans.**

**Q19.** What happens if you don't do something extra to treat an exception?

**Ans.** An exception object is created when a Python script raises an exception. If the script explicitly doesn't handle the exception, the program will be forced to terminate abruptly.The programs usually do not handle exceptions, and result in error messages.

**Q20.** What are your options for recovering from an exception in your script?

**Ans.**

Q21. Describe two methods for triggering exceptions in your script.

**Ans.**

Q22. Identify two methods for specifying actions to be executed at termination time, regardless of whether or not an exception exists.

**Ans.**

Q23. What is the purpose of the try statement?

**Ans.**

**Q24.** What are the two most popular try statement variations?

**Ans.**

**Q25.** What is the purpose of the raise statement?

**Ans.** The raise keyword is used to raise an exception. It raises an error and stops the control flow of the program.

Syntax: raise  {name\_of\_ the\_ exception\_class}

* It helps us raise exceptions when we may run into situations where execution can’t proceed.
* Raise allows us to throw one exception at any time.

**Q26.** What does the assert statement do, and what other statement is it like?

**Ans. Assertions** in any programming language are the debugging tools that help in the smooth flow of code. Assertion is the boolean expression that checks if the statement is True or False. If the statement is true then it continues the execution, but if the statement is False then it stops the execution of the program and throws an error.

Python has built-in *assert* statement to use assertion condition in the program.

***Syntax: assert condition, error\_message(optional)***

*assert* statement has a condition or expression which is supposed to be always true. If the condition is false assert halts the program and gives an *AssertionError*.

**Q27.** What is the purpose of the with/as argument, and what other statement is it like?

**Ans. with statement** is used in exception handling to make the code cleaner and much more readable. The with statement is a replacement for commonly used try/finally error-handling statements. The with statement automatically closes the file after you’ve completed writing it.

**Q28.** What are \*args, \*\*kwargs?

**Ans.** Special Symbols Used for passing arguments are:

* \*args (Non-Keyword Arguments)
* \*\*kwargs (Keyword Arguments)

\*args, which allows us to pass a variable number of non-keyword arguments to a function. Non-keyword here means that the arguments should not be a dictionary (key-value pair), and they can be numbers or strings. The syntax is to use the symbol \* to take in a variable number of arguments; by convention, it is often used with the word args. One thing to note here is that "args" is just an identifier. It can be named anything relevant.

#### \*\*kwargs allows us to pass any number of keyword arguments. In Python, these keyword arguments are passed to the program as a Python dictionary. Hence it maps each keyword to the value that we pass alongside it. Use double asterisk \*\* before the parameter name to denote this type of argument. The arguments are passed as a dictionary and these arguments make a dictionary inside function with name same as the parameter excluding double asterisk \*\*.

**Q29.** How can I pass optional or keyword parameters from one function to another?

**Ans.** The point to be taken care while passing without using keyword arguments is :

* The order in which parameters are defined in function should be maintained while calling the function.
* The values for the non-optional parameters should be passed otherwise it will throw an error.
* The value of the default arguments can be either passed or ignored.

In case of Passing keyword arguments, the parameters in functions are written in the form “datatype keyword-name”. So, the functions are called using the keyword name for passing the values. This relieves the programmer from learning the order in which the parameters are to be passed.There should be no difference between the passed and declared keyword names.

**Q30.** What are Lambda Functions?

**Ans.** In Python, a lambda function is a special type of function without the function name (anonymous functions). lambda keyword is used instead of def to create a lambda function. This function can have any number of arguments but only one expression, which is evaluated and returned.

Syntax: lambda arguments: expression

We can assign a lambda function to a variable and then call that variable as a normal function

**Q31.** Explain Inheritance in Python with an example?

**Ans.** Create a class named Person, with firstname, lastname properties, and a printname and details as method. To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class.

Create a class named Employee, which will inherit the properties and methods from the Person class.

Now Employee class has the same properties and methods as the Person class.

Use the Employee class to create an object, and then execute the details and printname method:

# Parent class

class Person:

  def \_\_init\_\_(self, fname, lname, age):

    self.firstname = fname

    self.lastname = lname

    self.age = age

  def printname(self):

    print(self.firstname, self.lastname)

  def details(self):

        print("My name is {}".format(self.firstname, self.lastname))

        print("Age: {}".format(self.age))

# Child class

class Employee(Person):

    def \_\_init\_\_(self, fname, lname, age, employeeid, salary, post):

        self.employeeid = employeeid

        self.salary = salary

        self.post = post

        Person.\_\_init\_\_(self, fname, lname, age)

    def details(self):

        print("My name is {}".format(self.firstname, self.lastname))

        print("Age: {}".format(self.age))

        print("Employee Id: {}".format(self.employeeid))

        print("Post: {}".format(self.post))

# object variable

a=Employee('Madhav', 'Das', 23, 4578, 15000, 'Intern')

a.printname()

a.details()

Output

*Madhav Das*

*My name is Madhav*

*Age: 23*

*Employee Id: 4578*

*Post: Intern*

**Q32.** Suppose class C inherits from classes A and B as class C(A,B).Classes A and B both have their own versions of method func(). If we call func() from an object of class C, which version gets invoked?

**Ans.**

**Q33.** Which methods/functions do we use to determine the type of instance and inheritance?

**Ans.** Two built-in functions **isinstance()** and **issubclass()** are used to check inheritances.

Syntax: [isinstance](https://www.geeksforgeeks.org/python-isinstance-method/)(object, classinfo)

The **isinstance()** function checks if the *object* (first argument) is an instance or subclass of the *classinfo* class (second argument). Classinfo can be class, type, or tuple of classes. For example, you can pass int, str, list, dict, or any user-created class.

The function isinstance() returns True if the object is an instance of the class or other classes derived from it.

 The **issubclass()** method asks whether one class is a subclass of another class.

### *Syntax: issubclass(class, classinfo)*

The issubclass() function checks if the class argument (first argument) is a subclass of classinfo class (second argument).

**Q34.**Explain the use of the 'nonlocal' keyword in Python.

### **Ans.** Based on the scope, we can classify Python variables into three types:

### Local Variables

### Global Variables

### Nonlocal Variables

### Nonlocal variables are used in nested functions whose local scope is not defined. This means that the variable can be neither in the local nor the global scope. We use the nonlocal keyword to create nonlocal variables. The nonlocal keyword is used in nested functions to reference a variable in the parent function.

**Q35.** What is the global keyword?

**Ans.** In Python, the keyword global allows us to modify the variable outside of the current scope, i.e., it helps to make changes to the variable in a local context. we can only access the global variable but cannot modify it from inside the function.

* When we create a variable inside a function, it is local by default.
* When we define a variable outside of a function, it is global by default. You don't have to use the global keyword.
* We use the global keyword to read and write a global variable inside a function.