**Python OOP Assignment**

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

It allows us to develop applications using an Object-Oriented approach. In Python, we can easily create and use classes and objects. An object-oriented paradigm is to design the program using classes and objects. The object is related to real-word entities such as book, house, pencil, etc.

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

An inheritance search looks for an attribute first in the instance object, then in the class the instance was created from, then in all higher superclasses, progressing from left to right (by default). The search stops at the first place the attribute is found.

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

A class is a type of blueprint that you can use to make objects. A concrete 'thing' that you constructed using a certain class is an object, which is an instance of a class. So, while the terms 'object' and 'instance' are interchangeable, the term 'instance' refers to an object's relationship to its class

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

Generally, when we call a method with some arguments, the corresponding class function is called by placing the method's object before the first argument. So, anything like **obj.meth(args)** becomes**Class.meth(obj, args).** The calling process is automatic while the receiving process is not (its explicit).

This is the reason the first parameter of a function in class must be the object itself. Writing this parameter as self is merely a convention. It is not a keyword and has no special meaning in Python. We could use other names (like this) but it is highly discouraged. Using names other than self is frowned upon by most developers and degrades the readability of the code

Q5. What is the purpose of the init method?

This method is called when an object is created from a class and it allows the class to initialize the attributes of the class.

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

In Python, a class can be created by using the keyword class, followed by the class name.

Q7. What is the process for creating a class?

In Python, a class can be created by using the keyword class, followed by the class name.

Q8. How would you define the superclasses of a class?

A class that is derived from another class is called a subclass (also a derived class, extended class, or child class). The class from which the subclass is derived is called a superclass (also a base class or a parent class).

Q9. What is the relationship between classes and modules?

Modules are collections of methods and constants. They cannot generate instances. Classes may generate instances (objects), and have per-instance state (instance variables).

Q10. How do you make instances and classes?

To create instances of a class, you call the class using class name and pass in whatever arguments its \_\_init\_\_ method accepts.

Q11. Where and how should be class attributes created?

**Class attributes** are the variables defined directly in the class that are shared by all objects of the class.

**Instance attributes** are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor.

Class attributes are the variables defined directly in the class that are shared by all objects of the class. Instance attributes are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor. Defined directly inside a class.

Q12. Where and how are instance attributes created?

Instance attributes are defined in the constructor. Defined directly inside a class. Defined inside a constructor using the self parameter. Shared across all objects

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

**self** refers to the instance whose method was called. **self** refers to the **class** that was inherited from to create the object using **self**

Q14. How does a Python class handle operator overloading?

The operator overloading in Python means provide extended meaning beyond their predefined operational meaning. Such as, we use the "+" operator for adding two integers as well as joining two strings or merging two lists. We can achieve this as the "+" operator is overloaded by the "int" class and "str" class.

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

When one or both operands are of a user-defined class or structure type, operator overloading makes it easier to specify user-defined implementation for such operations. This makes user-defined types more similar to the basic primitive data types in terms of behavior

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

The most frequent instance is the adding up operator '+', where it can be used for the usual addition and also for combining two different strings. As mentioned on top, the plus symbol's practice in dissimilar forms is the largest classic example of the operator level overloading process.

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

Both inheritance and polymorphism are fundamental concepts of object oriented programming. These concepts help us to create code that can be extended and easily maintainable.

Q18. Describe three applications for exception processing.

Exception handling is the process of responding to unwanted or unexpected events when a computer program runs. Exception handling deals with these events to avoid the program or system crashing, and without this process, exceptions would disrupt the normal operation of a program.

**It can consist of 3 steps:**

* a try block that encloses the code section which might throw an exception,
* one or more catch blocks that handle the exception and.
* a finally block which gets executed after the try block was successfully executed or a thrown exception was handled.

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

## if you don’t handle exceptions

When an exception occurred, if you don’t handle it, the program terminates abruptly and the code past the line that caused the exception will not get executed.

## Example

Generally, an array is of fixed size and each element is accessed using the indices. For example, we have created an array with size 7. Then the valid expressions to access the elements of this array will be a[0] to a[6] (length-1).

Whenever, you used an –ve value or, the value greater than or equal to the size of the array, then the **ArrayIndexOutOfBoundsException** is thrown.

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

The except block is executed when an exception occurs within the try block.

The optional else block is executed only if there were no exceptions after try and before finally.

The finally block contains instructions that are always executed at the end, regardless of whether exceptions occurred in the try block.

def test():

try:

# Do something

...

except Exception as e:

Log.Error(str(e))

else:

# Do something else if there were no errors

# You can handle other potential exceptions here:

try:

...

except Exception as e:

Log.Error(str(e))

...

finally:

# Finalize the run

Q21. Describe two methods for triggering exceptions in your script

In Python, exceptions can be handled by two new methods:

* Try: Catches exceptions raised by [Python](https://intellipaat.com/blog/tutorial/python-tutorial/what-is-python/) or a program
* Raise: A custom exception that triggers an exception manually

**Try-except-else Clause**

It gets initiated with a try header line which is followed by a block of indented statements and then by one or more optional except clauses and then at the end an optional else clause can be used as shown below:

try:

Your statements

except Exception\_1:

If there is Exception\_1 then execute this block- statement

except Exception\_2:

If there is Exception\_2 then execute this block-statement

else:

if no exception was raised-statement

Now, let us understand this with an example:

try:

fp = open(‘myfile.txt’)

line = f.readline()

i = int(s.strip())

except (IOError,ValueError) as e:

print (“check if the file is read-only.”,e.errno)

except:

print (“Unexpected error”)

chunk of code is written inside the try block that is suspected to raise a Python exception. Then, to handle that exception, we write the exception block.

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

### Finally Statement in Python

Finally block always executes irrespective of an exception being thrown or not. The final keyword allows you to create a block of code that follows a try-catch block.

Finally, clause is optional. It is intended to define clean-up actions which should be that executed in all conditions.

try:

raise KeyboardInterrupt

finally:

print 'welcome, world!'

Output

Welcome, world!

KeyboardInterrupt

Finally, clause is executed before try statement

Q23. What is the purpose of the try statement?

The try statement allows you to define a block of code to be tested for errors while it is being executed.

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

The Different Try/Except Variations. So far we've used a try / except and even a try / except / except , but this is only two-thirds of the story. There are two other optional segments to a try block: else and finally . Both of these optional blocks will come after the try and the except .

Q25. What is the purpose of the raise statement?

The raise statement allows you to force an error to occur. You can define both the type of error and the text that prints to the user. Note that the argument to raise must either be an exception instance or a subclass deriving from exception

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

The assert keyword is used when debugging code. The assert keyword lets you test if a condition in your code returns True, if not, the program will raise an AssertionError. You can write a message to be written if the code returns False, check the example below.

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

n Python, the with statement replaces a try-catch block with a concise shorthand. More importantly, it ensures closing resources right after processing them. A common example of using the with statement is reading or writing to a file. A function or class that supports the with statement is known as a context manager.

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

In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols:

1. \*args (Non Keyword Arguments)
2. \*\*kwargs (Keyword Arguments)

## Python \*args

As in the above example we are not sure about the number of arguments that can be passed to a function. Python has \*args which allow us to pass the variable number of non keyword arguments to function.

In the function, we should use an asterisk \* before the parameter name to pass variable length arguments.The arguments are passed as a tuple and these passed arguments make tuple inside the function with same name as the parameter excluding asterisk \*.

## Python \*\*kwargs

Python passes variable length non keyword argument to function using \*args but we cannot use this to pass keyword argument. For this problem Python has got a solution called \*\*kwargs, it allows us to pass the variable length of keyword arguments to the function.

In the function, we use the 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?

In Python, when we define functions with default values for certain parameters, it is said to have its arguments set as an option for the user. Users can either pass their values or can pretend the function to use theirs default values which are specified.

In this way, the user can call the function by either passing those optional parameters or just passing the required parameters.

There are two main ways to pass optional parameters in python

* Without using keyword arguments.
* By using keyword arguments.

## ****Passing without using keyword arguments****

* The order of parameters should be maintained i.e. 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.

## ****Passing with keyword arguments****

When functions are defined then the parameters are written in the form “datatype keyword-name”. So python provides a mechanism to call the function using the keyword name for passing the values. This helps the programmer by relieving them not to learn the sequence or the order in which the parameters are to be passed.

Some important points we need to remember are as follows:

* In this case, we are not required to maintain the order of passing the values.
* There should be no difference between the passed and declared keyword names

## Q30. What are Lambda Functions?

A lambda function is a small anonymous function. A lambda function can take any number of arguments, but can only have one expression.

Q31. Explain Inheritance in Python with an example?

Inheritance allows us to create a new class from an existing class.

The new class that is created is known as **subclass** (child or derived class) and the existing class from which the child class is derived is known as **superclass** (parent or base class).

Python Inheritence Syntax

# define a superclass

class super\_class:

# attributes and method definition

# inheritance

class sub\_class(super\_class):

# attributes and method of super\_class

# attributes and method of sub\_class

Here, we are inheriting the sub\_class class from the super\_class class.

**Example 1: Python Inheritance**

class Animal:

# attribute and method of the parent class

name = ""

def eat(self):

print("I can eat")

# inherit from Animal

class Dog(Animal):

# new method in subclass

def display(self):

# access name attribute of superclass using self

print("My name is ", self.name)

# create an object of the subclass

labrador = Dog()

# access superclass attribute and method

labrador.name = "Rohu"

labrador.eat()

# call subclass method

labrador.display()

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

I can eat

My name is Rohu

In the above example, we have derived a subclass Dog from a superclass Animal

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?

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

Using isinstance() function, we can test whether an object/variable is an instance of the specified type or class such as int or list. In the case of inheritance, we can checks if the specified class is the parent class of an object

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

The nonlocal keyword is used to work with variables inside nested functions, where the variable should not belong to the inner function. Use the keyword nonlocal to declare that the variable is not local.

Q35. What is the global keyword?

In Python, global keyword allows us to modify the variable outside of the current scope.

It is used to create a global variable and make changes to the variable in a local context.

Before we learn about global keyword, make sure you have got some basics of [Python Variable Scope](https://www.programiz.com/python-programming/global-local-nonlocal-variables).

First let's try to access a global variable from the inside of a function,

c = 1 # global variable

def add():

print(c)

add()

# Output: 1

Here, we can see that we have accessed a global variable from the inside of a function.

However, if we try to modify the global variable from inside a function as:

# global variable

c = 1

def add():

# increment c by 2

c = c + 2

print(c)

add()

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

UnboundLocalError: local variable 'c' referenced before assignment

This is because we can only access the global variable but cannot modify it from inside the function.