**Python OOP Assignment**

Q1. What is the purpose of Python's OOP?  
 Object-oriented programming (OOP) is a way of thinking about and organizing code for maximum reusability. With this type of programming, a program comprises objects that can interact with the user, other objects, or other programs. This makes programs more efficient and easier to understand.

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?

Class variables can only be assigned when a class has been defined. Instance variables, on the other hand, can be assigned or changed at any time. Both class variables and instance variables store a value in a program, just like any other Python variable.

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

self is the first method of every Python class  
  
When Python calls a method in your class, it will pass in the actual instance of that class that you're working with as the first argument. Some programming languages use the word this to represent that instance, but in Python we use the word self .

Q5. What is the purpose of the init method?

The \_\_init\_\_ method lets the class initialize the object's attributes and serves no other purpose. It is only used within classes.

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

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

"This would create first object of Employee class"

emp1 = Employee("Zara", 2000)

"This would create second object of Employee class"

emp2 = Employee("Manni", 5000)

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?

The difference between a class and a module in python is that 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.

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?

A class attribute is a Python variable that belongs to a class rather than a particular object. It's shared between all the objects of this class and is defined outside the constructor function, \_\_init\_\_(self,...) , of the 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?

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

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?

Ensures that objects of a class behave consistently with built-in types and other user-defined types. Makes it simpler to write code, especially for complex data types. Allows for code reuse by implementing one operator method and using it for other operators.

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.

Built-in Exceptions

|  |  |
| --- | --- |
| **Exception** | **Description** |
| IndentationError | Raised when indentation is not correct |
| IndexError | Raised when an index of a sequence does not exist |
| KeyError | Raised when a key does not exist in a dictionary |
| KeyboardInterrupt | Raised when the user presses Ctrl+c, Ctrl+z or Delete |

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.

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

* A single try statement can have multiple except statements. ...
* You can also provide a generic except clause, which handles any exception.
* After the except clause(s), you can include an else-clause. ...
* The else-block is a good place for code that does not need the try: block's protection.

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

To avoid such a scenario, there are two methods to handle Python exceptions: Try – This method catches the exceptions raised by the program. Raise – Triggers an exception manually using custom exceptions.

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

The `finally` keyword in the try-except block is always **executed**, **irrespective** of **whether** there is an **exception** or **not**

Q23. What is the purpose of the try statement?

The try block lets you test a block of code for errors. The except block lets you handle the error.

Q24. What are the two most popular try statement 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 keyword raises a specific exception when a condition is met or the code encounters an error. The exception raised by the program may either be an exception instance or an exception class. When you use the raise keyword, you can define what kind of error the machine should raise for a particular 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?

. Sometimes, we do not know in advance the number of arguments that will be passed into a function. To handle this kind of situation, we can use arbitrary arguments in Python.

Arbitrary arguments allow us to pass a varying number of values during a function call.

We use an asterisk (\*) before the parameter name to denote this kind of argument.

def find\_sum(\*numbers):

result = 0

for num in numbers:

result = result + num

print("Sum = ", result)

# function call with 3 arguments

find\_sum(1, 2, 3)

# function call with 2 arguments

find\_sum(4, 9)

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

When you don’t know in advance about the number of arguments to be passed, the arguments are variable-length. Include an asterisk i.e. \* before the parameter name while defining the function.

Let see an example:

def demo(\*car):

print("Car 1 = ",car[0])

print("Car 2 = ",car[1])

print("Car 3 = ", car[2])

print("Car 4 = ", car[3])

# call

demo("Tesla", "Audi", "BMW", "Toyota")

### **Output**

('Car 1 = ', 'Tesla')

('Car 2 = ', 'Audi')

('Car 3 = ', 'BMW')

('Car 4 = ', 'Toyota')

When you don’t know in advance about the number of keyword arguments to be passed, the arguments are arbitrary keyword arguments.

### **Example**

Let us see an example −

def demo(\*\*c):

print("Car Name: "+c["name"])

print("Car Model: "+c["model"])

# call

demo(name = "Tesla", model = "2022")

### **Output**

Car Name: Tesla

Car Model: 2022

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

To pass, collect the arguments using the \* and \*\* in the function’s parameter list. Through this, you will get the positional arguments as a tuple and the keyword arguments as a dictionary. Pass these arguments when calling another function by using \* and \*\* −

def f(a, \*args, \*\*kwargs):

...

kwargs['width'] = '14.3c'

...

g(a, \*args, \*\*kwargs)

Q30. What are Lambda Functions?

lambda is a keyword in Python for defining the anonymous function. argument(s) is a placeholder, that is a variable that will be used to hold the value you want to pass into the function expression. A lambda function can have multiple variables depending on what you want to achieve.

Q31. Explain Inheritance in Python with an example?

The concept of inheriting the properties from one class into multiple classes separately is known as hierarchical inheritance.

Example:

class x(object):   
def m1(self):   
print("in m1 of x")  
class y(x):   
def m2(self):   
print("in m2 of y")  
class z(x):   
def m3(self):   
print("in m3 of z")  
y1=y()  
y1.m1()  
y1.m2()  
a=y1.--hash--()  
print(a)  
z1=z()  
z1.m1()  
z1.m3()  
b=z1.hash--()  
print(b)

Output:

M m1 of XIn m2 of Y2337815In m1 of XIn m3 of Z2099735

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?

On Multiple Inheritance in Python, we discussed Method Resolution Order (MRO). C does not contain its own version of func(). Since the interpreter searches in a left-to-right fashion, it finds the method in A, and does not go to look for it in B.

Q33. Which methods/functions do we use to determine the type of i  Here, we talk about three methods/functions- type(), isinstance() and issubclass().

a. type(): This tells us the type of object we're working with.

type(3)

<class ‘int’>

type(False)

<class ‘bool’>

type(lambda :print("Hi"))

<class ‘function’>

type(type)

<class ‘type’>

b. isinstance()

This takes in two arguments- a value and a type. If the value is of the kind of the specified type, it returns True. Else, it returns False.

isinstance(3,int)

True

isinstance((1),tuple)

False

isinstance((1,),tuple)

True

c. issubclass()

This takes two classes as arguments. If the first one inherits from the second, it returns True. Else, it returns False.

class A: pass class B(A): pass issubclass(B,A)

True

issubclass(A,B)

False

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?

Global keyword is used when we want to read or write any global variable value inside the function. The global keyword used for a variable declared outside the function does not have any effect on it. In the same line, a variable cannot be declared global and assigned a value. E.g. global x = 5 is not allowed.