**Python OOP Assignment**

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

In Python, object-oriented **Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming**. **Object-oriented programming** (OOP) is a method of structuring a program by bundling related properties and behaviors into individual **objects**. In this tutorial, you’ll learn the basics of object-oriented programming in Python.

**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 object is a template or blueprint that defines the properties and behavior of objects that belong to the class. It is an abstract representation of a group of objects. An instance object is an individual object created from a class. It is a concrete object that has its own identity, state, and behavior.

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

The first argument in a class's method function is special because it is used to refer to the instance object, also known as the 'self' argument. This argument is used to access the data and methods associated with the particular instance object.

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

Whenever object-oriented programming is done in Python, we mostly come across **\_\_init\_\_ method in oops** which we usually don’t fully understand.  The examples above are classes and objects in their simplest form, and are not really useful in real life applications.To understand the meaning of classes we have to understand the built-in \_\_init\_\_() function.All classes have a function called \_\_init\_\_(), which is always executed when the class is being initiated.Use the \_\_init\_\_() function to assign values to object properties, or other operations that are necessary to do when the object is being created.

**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.

For example: - "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?**

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. A Class is like an object constructor, or a "blueprint" for creating objects. To create a class, use the keyword “class”: For example: class Person, class Employee, class Car, etc.

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

The class whose properties gets inherited by another class is known as **superclass or parent class** and the class which inherits the properties of another class is known as the **subclass**. A subclass inherits all data and behavior of parent class. But we can also add more information and behavior to the subclass and also override its behavior. The super() function is used to give access to methods and properties of a parent or sibling class. The super() function returns an object that represents the parent class.

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

So a module in python is simply a way to organize the code, and it contains either python classes or just functions. If you need those classes or functions in your project, you just import them. For instance, the math module in python contains just a bunch of functions, and you just call those needed.

**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**.** Object of the Class: Declare an object of your class in the main method or from outside the class. Method Definition: write the method's header and body code like below. Method Call: whenever you want to use the method, call objectName.methodName().

**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.

The following table lists the difference between class attribute and instance attribute:

| Class Attribute | Instance Attribute |
| --- | --- |
| Defined directly inside a class. | Defined inside a constructor using the self parameter. |
| Shared across all objects. | Specific to object. |
| Accessed using class name as well as using object with dot notation, e.g. classname.class\_attribute or object.class\_attribute | Accessed using object dot notation e.g. object.instance\_attribute |
| Changing value by using classname.class\_attribute = value will be reflected to all the objects. | Changing value of instance attribute will not be reflected to other objects. |

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

An instance attribute is a Python variable belonging to one, and only one, object. This variable is only accessible in the scope of this object, and it's defined **inside the constructor function, \_\_init\_\_(self,..) of the class**.

Instance attribute created under the init function;

For example:- class Employee:

def \_init\_(**self,** **name, age**)

self.name = name;

self.age = age;

**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. It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class. “self” is the current object, that is declare in the \_init\_ function.

**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. The user can notice that the same inbuilt operator or function is showing different behaviour for objects of different classes. This process is known as operator overloading.

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

Consider that we have two objects which are a physical representation of a class (user-defined data type) and we have to add two objects with binary ‘+’ operator it throws an error, because compiler don’t know how to add two objects. So we define a method for an operator and that process is called operator overloading. We can overload all existing operators but we can’t create a new operator. To perform operator overloading, Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator. For example, when we use + operator, the magic method \_\_add\_\_ is automatically invoked in which the operation for + operator is defined.  
Overloading binary + operator in Python :  
When we use an operator on user defined data types then automatically a special function or magic function associated with that operator is invoked. Changing the behavior of operator is as simple as changing the behavior of method or function. You define methods in your class and operators work according to that behavior defined in methods. When we use + operator, the magic method \_\_add\_\_ is automatically invoked in which the operation for + operator is defined. There by changing this magic method’s code, we can give extra meaning to the + operator.

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

A very popular and convenient example is the Addition (+) operator.

Just think how the ‘+’ operator operates on two numbers and the same operator operates on two strings. It performs “Addition” on numbers whereas it performs “Concatenation” on strings.

Operators in Python work for built-in classes, like int, str, list, etc. But you can extend their operability such that they work on objects of user-defined classes too.

**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.Inheritance is a great way to eliminate unnecessary repetitive code. A child class can inherit from the parent class partially or entirely. Python is quite flexible with regards to inheritance. We can add new attributes and methods as well as modify the existing ones. Polymorphism contributes to Python’s flexibility as well. An object with a particular type can be used as if it belonged to a different type. We have seen an example of it with the give\_raise method.

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

|  |
| --- |
|  |
| 16 | **KeyError**  Raised when the specified key is not found in the dictionary. |
| 17 | **NameError**  Raised when an identifier is not found in the local or global namespace. |
| 18 | **UnboundLocalError**  Raised when trying to access a local variable in a function or method but no value has been assigned to it. |
| 19 | **EnvironmentError**  Base class for all exceptions that occur outside the Python environment. |

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

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 as shown here:

Type Error

a = 2

b = 'DataCamp'

a + b

TypeError Traceback (most recent call last)

<ipython-input-7-86a706a0ffdf> in <module>

1 a = 2

2 b = 'DataCamp'

----> 3 a + b

TypeError: unsupported operand type(s) for +: 'int' and 'str'

Zero Division Error

100 / 0

---------------------------------------------------------------------------

ZeroDivisionError Traceback (most recent call last)

<ipython-input-43-e9e866a10e2a> in <module>

----> 1 100 / 0

ZeroDivisionError: division by zero

There are various types of Python exceptions, and the type is printed as part of the message: the types in the above two examples are ZeroDivisionError and TypeError. Both the error strings printed as the exception type is the name of the Python's built-in exception.

The remaining part of the error line provides the details of what caused the error based on the type of exception.

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

You can also **provide a generic except clause**, which handles any exception. After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception. 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:

1. **Try**– This method catches the exceptions raised by the program
2. **Raise**– Triggers an exception manually using custom exceptions

Let’s start with the try statement to handle exceptions. Place the critical operation that can raise an exception inside the try clause. On the other hand, place the code that handles the exceptions in the except clause.

Developers may choose what operations to perform once it catches these exceptions. Take a look at the sample code below:

import sys

list = [‘x’, 1e-15, 5]

for result in list:

    try:

        print(“The result is”, result)

        y = 1/int(result)

        break

    except:

        print(“Whew!”, sys.exc\_info()[0], “occurred.”)

        print(“Next input please.”)

        print()

print(“The answer of”, result, “is”, y)

The program has an array called list with three elements. Next, the line that causes an exception is placed inside the try block. If there are no exceptions, the except block will skip, and the logic flow will continue until the last element. However, if an exception occurs, the except block will catch it.

**The output**:

The result is x

Whew! <class ‘ValueError’> occurred.

Next input, please.

The result is 1e-15

Whew! <class ‘ZeroDivisionError’> occurred.

Next input, please.

The result is 5

The answer to 5 is 0.2

To print the name of the exception, we use the exc\_info()function inside the sys module. The first two elements cause exceptions since an integer can’t be divided with string and a zero value. Hence, it raised *ValueError* and *ZeroDivisionError* exceptions.

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

**Using the finally block in a try-catch-finally structure in Java or similar constructs in other programming languages.**

**Implementing a shutdown hook in Java, which allows registering a thread to be executed when the JVM is shutting dow**n.

**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. Also, there’s nothing stopping you from using both else and finally in a single statement — but keep them in that order if you do.

Let’s go through each individually and see how they extend the behavior of a simple try/except.

Try/Except/Else

When attaching an else statement to the end of a try/except, this code will be executed after the try has been completed, but only if no exceptions occur.

We can take the previous example of prompting a user for an integer input and use an else block to thank them for valid input and breaking out of the while loop.

Try/Except/Finally

When attaching a finally statement to the end of a try/except, this code will be executed after the try has been completed, regardless of exceptions.

Again, we’ll use our previous example and add a simple counter to illustrate this behavior.

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

The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

**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?**

The with statement is a replacement for commonly used try/finally error-handling statements. A common example of using the with statement is opening a file.

To open and write to a file in Python, you can use the with statement as follows:

with open("example.txt", "w") as file:

file.write("Hello World!")

The with statement automatically closes the file after you’ve completed writing it.

Under the hood, the with statement replaces this kind of try-catch block:

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

The special syntax **\*args** in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-key worded, variable-length argument list.

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.

What \*args allows you to do is take in more arguments than the number of formal arguments that you previously defined. With \*args, any number of extra arguments can be tacked on to your current formal parameters (including zero extra arguments).

For example, we want to make a multiply function that takes any number of arguments and is able to multiply them all together. It can be done using \*args.

Using the \*, the variable that we associate with the \* becomes an iterable meaning you can do things like iterate over it, run some higher-order functions such as map and filter, etc.

The special syntax **\*\*kwargs** in function definitions in python is used to pass a keyworded, variable-length argument list. We use the name kwargs with the double star. The reason is that the double star allows us to pass through keyword arguments (and any number of them).

A keyword argument is where you provide a name to the variable as you pass it into the function.

One can think of the kwargs as being a dictionary that maps each keyword to the value that we pass alongside it. That is why when we iterate over the kwargs there doesn’t seem to be any order in which they were printed out.

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

To pass optional or keyword parameters from one function to another, collect the arguments using the \* and \*\* specifiers in the function’s parameter list But, at first, do know what are \*args and \*\*args in Python. Let us understand them −

**Variable-length/ Arbitrary arguments in Python (\*args)**

Example

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 us 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')

**Arbitrary Keyword Arguments in Python (\*\*kwargs)**

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

**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 \*\* −

**Q30. What are Lambda Functions?**

Python Lambda Functions are anonymous function means that the function is without a name. As we already know that the def keyword is used to define a normal function in Python. Similarly, the lambda keyword is used to define an anonymous function in Python.

Python Lambda Function Syntax

Syntax: lambda arguments: expression

This function can have any number of arguments but only one expression, which is evaluated and returned.

One is free to use lambda functions wherever function objects are required.

You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.

It has various uses in particular fields of programming, besides other types of expressions in functions.

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

One of the core concepts in object-oriented programming (OOP) languages is inheritance. It is a mechanism that allows you to create a hierarchy of classes that share a set of properties and methods by deriving a class from another class. Inheritance is the capability of one class to derive or inherit the properties from another class.

**Benefits of inheritance are:**

* It represents real-world relationships well.
* It provides the **reusability** of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
* It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.
* Inheritance offers a simple, understandable model structure.
* Less development and maintenance expenses result from an inheritance.

**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?**

The function which declare in Class A and Class B that can be invoked when we call function from an object of class C.

**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. For example, isinstance(x, int) to check if x is an instance of a class int.

* se isinstance() to check an instance's type: isinstance(obj, int) will be True only if obj.\_\_class\_\_ is int or some class derived from int .
* Use issubclass() to check class inheritance: issubclass(bool, int) is True since bool is a subclass of int .

**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, the 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 the global keyword, make sure you have got some basics of Python Variable Scope.

A global keyword is a keyword that allows a user to modify a variable outside the current scope. It is used to create global variables in Python from a non-global scope, i.e. inside a function. Global keyword is used inside a function only when we want to do assignments or when we want to change a variable. Global is not needed for printing and accessing.

Rules of global keyword:

If a variable is assigned a value anywhere within the function’s body, it’s assumed to be a local unless explicitly declared as global.

Variables that are only referenced inside a function are implicitly global.

We use a global keyword to use a global variable inside a function.

There is no need to use global keywords outside a function.

Use of global keyword in Python: To access a global variable inside a function, there is no need to use a global keyword.