**Python OOP Assignment**

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

Ans) 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.

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

Ans) 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?

Ans) **Class object:**

when we create a class in python then a class object is created so whenever python finds a class statement in the whole program then it creates a class object and assigns a name to that object i.e. class name. As we know in python, everything is an object so the class itself is an object and is the instance of [metaclasses](https://artificialintelligencestechnology.com/python/metaclasses-in-python/).

Syntax:

class MyClass:

 pass

**Instance object:**

when we call a class, it creates an instance object of that class from which the object has been created. For example when we call the above-created class then it will create an instance object like this.

Syntax:

Obj1=MyClass()

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

Ans) The first argument of a method is called **self** . This is nothing more than a convention: the name self has no special meaning to Python. Self is the first argument to be passed in Constructor and Instance Method. Self must be provided as a First parameter to the Instance method and constructor. If you don't provide it, it will cause an error.

Q5. What is the purpose of the init method?

Ans) 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?

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

**Syntax:**

emp1 = Employee(“Varshi”, 50000)

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.

**Syntax:**

class = Car:

#statements

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

Ans) The class from which a class inherits is called the parent or superclass. A class which inherits from a superclass is called a subclass, also called heir class or child class. Superclasses are sometimes called ancestors as well.

**Syntax:**

class DerivedClassName(BaseClassName):

pass

**Example:**

class A:

pass

class B(A):

pass

class C(B):

pass

x = C()

print(isinstance(x, A))

Q9. What is the relationship between classes and modules?

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

Q10. How do you make instances and classes?

Ans) **Creating Instances:**

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

Syntax:

Emp1 = Employee(“Varshi”, 2000)

**Creating class:**

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

**Syntax:**

class = Car:

#statements

Q11. Where and how should be class attributes created?

Ans) To define a class attribute, you place it outside of the \_\_init\_\_() method. Use class\_name.class\_attribute or object\_name.class\_attribute to access the value of the class\_attribute.

**Syntax:**

class Test:

x = 10

def \_\_init\_\_(self):

self.x = 20

test = Test()

print(test.x) # 20

print(Test.x) # 10

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. Instance attributes are created inside the \_\_init\_\_ function.

**Syntax:** object.instance\_attribute

**Example:**

class Student:

school = "freeCodeCamp.org"

def \_\_init\_\_(self, name, course):

#Setting instance attributes

self.name = name

self.course = course

#Creating instance of a class

Student1 = Student("Jane", "JavaScript")

Student2 = Student("John", "Python")

#calling instance attributes

print(Student1.name) # Jane

print(Student2.name) # John

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

Ans) 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?

Ans) 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.

**Example:**

Python program for simply using the overloading operator for adding two objects.

class example:

    def \_\_init\_\_(self, X):

        self.X = X

    # adding two objects

    def \_\_add\_\_(self, U):

        return self.X + U.X

object\_1 = example( int( input( print ("Please enter the value: "))))

object\_2 = example( int( input( print ("Please enter the value: "))))

print (": ", object\_1 + object\_2)

object\_3 = example(str( input( print ("Please enter the value: "))))

object\_4 = example(str( input( print ("Please enter the value: "))))

print (": ", object\_3 + object\_4)

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

Ans) Operator overloading is mostly useful when you're making a new class that falls into an existing "Abstract Base Class" (ABC) -- indeed, many of the ABCs in standard library module collections rely on the presence of certain special methods (and special methods, one with names starting and ending with double underscores.

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

Ans) 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.

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

Ans) 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.

Ans) **Applications for exception processing:**

1. An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
2. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception.
3. An exception is a Python object that represents an error. In Python, we catch exceptions and handle them using try and except code blocks.

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

Ans) 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?

Ans) 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.

Ans) 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.

Ans) Finally block always executes irrespective of an exception being thrown or not. The final keywords allow to create a block of code which is mandatory in terms of execution.

Q23. What is the purpose of the try statement?

Ans) The try block lets you test a block of code for errors.

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

Ans) 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?

Ans) 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?

Ans) 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.

**Example:**

x = "hello"

#if condition returns False, AssertionError is raised:

assert x == "ok", "x should be 'hello'"

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

Ans) In Python, **with statement** is used in exception handling to make the code cleaner and much more readable. It simplifies the management of common resources like file streams.

**Syntax:**

# using with statement

with open('file\_path', 'w') as file:

file.write('hello world !')

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

Ans) \*args specifies the number of non-keyworded arguments that can be passed and the operations that can be performed on the function in Python whereas \*\*kwargs is a variable number of keyworded arguments that can be passed to a function that can perform dictionary operations.

**Example:**

#Non-key valued arguments

def example\_nonkeyed\_args(\*argv):

    for param in argv:

        print(param)

example\_nonkeyed\_args('hey', 'hii',4,5,5678.45678)

#key value type of arguments in python

def example\_of\_kwargs(\*\*kwargs):

    print("Value of host is: ", kwargs['host'])

    for k, v in kwargs.items():

        print("Key is: ", k, "value is: ",v)

example\_of\_kwargs(host='170.80.80.80', port=9021, pwd='DFJHMNB')

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

Ans) 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 \*\*.

**Example:**

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

...

kwargs['width'] = '14.3c'

...

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

Q30. What are Lambda Functions?

Ans) 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.

**Syntax:**

str1 = 'GeeksforGeeks'

# lambda returns a function object

rev\_upper = lambda string: string.upper()[::-1]

print(rev\_upper(str1))

Q31. Explain Inheritance in Python with an example?

Ans) Inheritance relationship defines the classes that inherit from other classes as derived, subclass, or sub-type classes. Base class remains to be the source from which a subclass inherits.

**Example:**

class Country:

     def ShowCountry(self):

         print(“This is Spain”);

class State(Country):

     def ShowState(self):

         print(“This is State”);

st =State();

st.ShowCountry();

st.ShowState();

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) Python will always call the first thing that is found in the method resolution order. Since you specified the inheritance as A, B then A will be found first, so its method will be called.

**Example:**

class A():

def func(self):

print('in A')

class B():

def func(self):

print('in B')

class C(A, B):

pass

C().func() # prints 'in A'

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

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

Ans) 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.

**Example:**

def myfunc1():  
  x = "John"  
  def myfunc2():  
    nonlocal x  
    x = "hello"  
  myfunc2()  
  return x  
  
print(myfunc1())

#Output: Hello

Q35. What is the global keyword?

Ans) 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.

**Example:**

# global variable

c = 1

def add():

# use of global keyword

global c

# increment c by 2

c = c + 2

print(c)

add()