1. OOPs (Object-Oriented Programming) – 10 Questions

1. What is the difference between self and cls in Python classes?

 self-> refers to instance of a class i.e object of object,not the class itself. It is used in instance methods.

cls-> refers to class itself. It is used in class methods

2. How does inheritance work in Python? Give an example with method overriding.

Inheritance means inherit /take the properties or features of the parent class by the derived class.

Method overriding is done when the child class defines a method with the same name as that of parent.

class Animal:

    def speak(self):

        print("Animal speaks")

class Dog(Animal):

    def speak(self):

        print("Dog barks")

d = Dog()

d.speak()

o/p: Dog Barks

3. What is method overloading in Python? Is it supported natively?

Method overloading is not supported in python.

Method Overloading means to have multiple methods with the same name but different parameters (like in Java or C++)

4. Define constructor and destructor in Python. When are they called?

\_\_init\_\_() is the constructor, which runs when an object is created.

\_\_del\_\_() is the destructor, which runs when the object is about to be destroyed/deleted.

5. What is the difference between instance method, class method, and static method?

* Instance: uses self() and it  has access to instance variables.
* Class: uses @classmethod works with cls() modifies class-level data.
* Static: uses @staticmethod and provides no access to class or instance directly.

 6. How do you restrict access to class attributes in Python (pseudo-private)?

Python does not have true private access modifiers.

Using double underscores we can restrict access. (\_\_var) is used for name mangling to pseudo-private, that means hiding the attributes. It is done in encapsulation

Using Single Underscore Prefix marks an attribute as protected.only still accessible from outside the class.

Makes it harder to access accidentally, but still not truly private.

 7. Write a Python class to demonstrate encapsulation with getter/setter methods.

class Person:

    def \_\_init\_\_(self):

           self.\_\_marks = 0

    def set\_marks(self, m):

           self.\_\_marks = m

    def get\_marks(self):

           return self.\_\_marks

8. What is polymorphism in Python? Show it with two unrelated classes using the same method name.

Polymorphism means having the same method or function name but it behaves differently or has/performs different features

class Fish:

    def swim(self):

        print("Fish swim")

class Human:

    def swim(self):

        print("Human swim")

for obj in (Fish(), Human()):

    obj.swim()\

o/p:

Fish swim

Human swim

9. What is a magic method? Name a few commonly used ones and their purpose.

Special methods with \_\_double underscores\_\_  and define how objects behave with operators are called magic methods.some of them are:

\_\_init\_\_

\_\_str\_\_

\_\_len\_\_

\_\_add\_\_

10. How do you use isinstance() and issubclass() functions?

isinstance()-> checks if the object (obj)is an instance of Class.

issubclass()-> it checks if subclass defined(sub) is a subclass of class

class A:

       print(“hi”)

class B(A):

     print(“hello”)

obj = B()

print(isinstance(obj, B))

print(isinstance(obj, A))

print(issubclass(B, A))

print(issubclass(A, B))

2. Decorators – 10 Questions

1. What is a decorator in Python and what is its typical use case?

Decorator in python means a function that modifies/enhances another function without changing its code.

2. Write a simple decorator that logs when a function is called.

def outer(func):

   def inner():

       print("before")

       func()

       print("after")

   return inner

@outer

def say\_hello():

   print("hello")

say\_hello()

o/p:

before

hello

after

3. Can you apply more than one decorator to a function? In what order are they applied?

yes.

def inner(func):

   def wrapper(\*args, \*\*kwargs):

       print("decorator1")

       return func(\*args, \*\*kwargs)

   return wrapper

def outer(func):

   def wrapper(\*args, \*\*kwargs):

       print("decorator2")

       return func(\*args, \*\*kwargs)

   return wrapper

@inner

@outer

def greet():

   print("Hi,Abhi!")

greet()

o/p:

decorator1

decorator2

Hi,Abhi!

4. What is the use of functools.wraps() in a decorator?

It preserves the metadata like name of the original function.

from functools import wraps

def logger(func):

   @wraps(func)

   def wrapper(\*args, \*\*kwargs):

       print(f"Calling {func.\_\_name\_\_}")

       return func(\*args, \*\*kwargs)

   return wrapper

@logger

def hello():

   print("Hello!")

print(hello.\_\_name\_\_)

o/p:hello

5. Convert the following decorator to one that accepts arguments (parameterized decorator).

def num(n):

   def decorator(func):

       def wrapper(\*args, \*\*kwargs):

           for i in range(n):

               func(\*args, \*\*kwargs)

       return wrapper

   return decorator

@num(5)

def welcome():

   print("Hi Students!")

welcome()

o/p:

Hi Students!

Hi Students!

Hi Students!

Hi Students!

Hi Students!

6. How can you write a decorator to check if the user is logged in before accessing a function? 

def login(func):

    def wrapper(user):

        if user.is\_logged\_in:

            return func(user)

        else:

            print("Please log in.")

    return wrapper

7. How does the @property decorator work? Give an example.

class Student:

    def \_\_init\_\_(self, name):

        self.\_name = name

    @property

    def name(self):

        return self.\_name.upper()

s = Student("abhi")

print(s.name)

o/p:ABHI

@property decorator gives us the property of the given or specified object

Ex: name can be converted to upper or lower case

And in shapes ex circle property will be area or radius of circle.

8. Write a decorator that catches and logs any exceptions in a function.

def divide(func):

    def wrapper(a, b):

        try:

            return func(a, b)

        except ZeroDivisionError as zde:

            return "Error,number can’t be divided by zero!"

    return wrapper

@divide

def divide(x, y):

    return x / y

print(divide(10, 2))

print(divide(10, 0))

9. What is the difference between function decorator and class decorator?

Function Decorator

Can be applied to Functions or methods

It is used to modify or extend a function's behavior without changing its code

A function object is taken as input and output is function modified

Class Decorator

Can be applied to only Classes

They are used to modify class behavior, inject methods or attributes, enforce constraints, register types

A class object is taken as input and output is wrapped class

10. Can decorators be used with class methods or static methods?

Yes they can be used with class methods and static methods

Decorators used for class method is : @classmethod

Decorators used for static method is: @staticmethod

3. Generators – 10 Questions

1. What is a generator function? How is it different from a normal function?

A generator function uses yield() to return values one at a time, iterate one at a time.

Does lazy iteration or execution.

Return suddenly returns the end value, but yield pauses execution and does’nt end the function.

2. Write a generator function to yield even numbers up to 20.

def even():

    for i in range(0, 21, 2):

        yield i

for num in even():

    print(num, end=" ")

o/p:0 2 4 6 8 10 12 14 16 18 20

3. What happens if you call next() on an exhausted generator?

When we call next() it raises a StopIteration exception.

num = (x for x in range(2))

print(next(num))

print(next(num))

print(next(num))

o/p:

0

1

Traceback (most recent call last):

  File "/Users/abhigna/PycharmProjects/PythonProject/day 9 python.py", line 4, in <module>

    print(next(num))

StopIteration

4. What is the use of yield? How does it help in memory efficiency?

 It pauses the function and returns a value without holding all results in memory.

 It helps to process large data because it performs line by line iteration or one iteration at a time(one piece at a time)

5. How do you use a generator expression? How is it different from list

Comprehension?

gen = (x\*x for x in range(5))

print(next(gen))

print(next(gen))

->generator is lazy and returns immediate value, we use ( ) for generators

->wheres in list it is not lazy and does not return immediate value ,we use [ ] for list

 6. Convert a normal function that returns a list into a generator.

Normal function:

def squares():

    return [x\*\*2 for x in range(5)]

print(squares())

generator():

def squares\_gen():

    for x in range(5):

        yield x\*\*2

print(list(squares\_gen()))

o/p:[ 0, 1, 4, 9, 16]

7. How would you read a large file using a generator to process it line by line?

def read(name):

    with open(“file.txt”,’r’) as file:

        for line in file:

            yield line.strip()

8. How does the generator maintain its state between calls?

The generator goes to  the last yield statement and resumes from there on the next callIt uses next() for this.

def simple\_generator():

   yield 1

   yield 2

gen = simple\_generator()

print(next(gen))

print(next(gen))

9. What is the difference between return and yield inside a function?

Return ends the function

yield pauses the function and can resume it(that is to continue from same point), then returns the stored value, it is used in generator.

def simple\_generator():

   yield 1

   yield 2

   yield 3

gen = simple\_generator()

print(next(gen))

print(next(gen))

print(next(gen))

def generator():

   return 1

  # return 2

  # return 3

print(generator()

o/p: yield returns

1

2

3

Returns gives only

1

10. What is the output of list(generator\_function()) and how does it differ from a list-returning function?

def gen():

    yield 1

    yield 2

    yield 3

print(list(gen()))

o/p:[1,2,3]

list(gen()) takes value from generator and stores in a list

Whereas in list-returning function it does not store a value, returns a list directly,because there is no generator. This is the major difference

4. Iterators – 10 Questions

1. What is the difference between an iterable and an iterator?

Iterable is an object you can loop over using for or while/do-while loop (ex:list).

Iterator is an object that produces items one at a time, iter one after other using \_\_next\_\_().

2. How do you make a class iterable using \_\_iter\_\_() and \_\_next\_\_()?

class countthree:

   def \_\_init\_\_(self):

       self.num=0

   def \_\_iter\_\_(self):

       return self

   def \_\_next\_\_(self):#magic methods

       if self.num>3:

           raise StopIteration

       current=self.num

       self.num+=1

       return current

it = countthree()

for val in it:

   print(val,end=" ")

 3. Explain what happens when StopIteration is raised.

It tells that loop has ended/or tells that it is the end of iteration.

it = iter([1])

print(next(it))

print(next(it))

o/p:

1

Traceback (most recent call last):

  File "/Users/abhigna/PycharmProjects/PythonProject/day 9 python.py", line 5, in <module>

    print(next(it))

StopIteration

4. Give an example of using iter() with a sentinel value.

data = iter(input, 'exit')

for line in data: print(line)

5. How does a for loop work internally with iterators?

First it calls \_\_iter\_\_() on the iterable

Then calls \_\_next\_\_()  repetedly

Then the for loop stops on Stop iteration, this occurs when the loop has ended or indicated its the end of iteration

6. What built-in functions rely on iterators (e.g., map, zip, filter)?

Map,zip,filer,enumerate these are few of the built in functions, all give iterators.

print(list(map(str, [1, 2, 3,4,5])))

print(list(zip("python", [1, 2, 3,4,5])))

print(list(filter(lambda x: x%2==0, [1,2,3,4])))

o/p:

['1', '2', '3', '4', '5']

[('p', 1), ('y', 2), ('t', 3), ('h', 4), ('o', 5)]

[2, 4]

7. How to manually loop over an iterator using next()?

num=[1,2,3]

it=iter(num)

print(next(it)) #->1

print(next(it)) # ->2

print(next(it))  #->3

8. Write a custom iterator that returns square of numbers from 1 to 5.

class Square:

   def \_\_init\_\_(self):

       self.num = 1

   def \_\_iter\_\_(self):

       return self

   def \_\_next\_\_(self):

       if self.num <= 5:

           result = self.num \*\* 2

           self.num += 1

           return result

       else:

           raise StopIteration

for n in Square():

   print(n)

o/p:

1

4

9

16

25

9. What happens when you try to iterate over an already exhausted iterator?

If we are not using next , we won’t get any output but if we use next() then,it gives error as Stop Iteration.

it = iter([1, 2])

list(it)

list(it)

o/p: [ ]

Using next()

num=[1,2]

it=iter(num)

print(next(it))

print(next(it))

print(next(it))

o/p:

Traceback (most recent call last):

  File "/Users/abhigna/PycharmProjects/PythonProject/day 9 python.py", line 7, in <module>

    print(next(it))

StopIteration

1

2

10. What is the use of the itertools module in python with itertools.

Itertools gives us efficient tools to work with iterators.

Count,chain,product,permutation,cycle

from itertools import count,cycle,product

for i in count(5):

   if i>8:

       break

   print(i,end=" ")

count=0

for x in cycle(["a","b"]):

   if count>3:

       break

   print(x,end=" ")

   count+=1

for item in product(["a", "b"], [1, 2]):

       print(item)