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

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

self refers to the instance of the class. cls refers to the class itself. self is used in instance methods; cls is used in class methods.

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

class Animal:

def speak(self):

return "Animal speaks"

class Dog(Animal):

def speak(self):

return "Dog barks"

d = Dog()

print(d.speak())

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

Python doesn't support method overloading natively like Java.

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

Constructor: \_\_init\_\_(), it is called when object is created.

Destructor: \_\_del\_\_(), it is called when object is deleted.

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

Instance Method: Uses self, accesses instance data.

Class Method: Uses cls, defined with @classmethod.

Static Method: No self or cls, uses @staticmethod.

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

Use underscores: \_var for protected, \_\_var for private.

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

class Person:

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

self.\_\_name = name

def get\_name(self):

return self.\_\_name

def set\_name(self, name):

self.\_\_name = name

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

**Polymorphism** means "many forms". It allows objects of different classes to be treated through the same interface — even if they behave differently.

Class Animal:

Def breed(self):

Print(“dog”)

Class Pet\_Animal(Animal):

Def breed(self):

Print(“Cat”)

Obj1 = Animal()

Obj1.breed()

Obj2 = Pet\_Animal()

Obj2.breed()

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

**Magic methods** is also called **dunder methods**, short for **"double underscore"** are special predefined methods in Python that start and end with double underscores

\_\_init\_\_: Constructor

\_\_str\_\_: String representation

\_\_len\_\_: Length

\_\_add\_\_: Addition

\_\_eq\_\_: Equality

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

isinstance(obj, Class) : checks if object belongs to a class.

issubclass(SubClass, Class) :checks if a class is a subclass.

2. Decorators – 10 Questions

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

A function that takes another function and extends or modifies its behavior without changing its source.

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

def log(func):

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

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

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

return wrapper

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

Yes, they are applied from bottom to top:

@decorator1

@decorator2

def func():

pass

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

Preserves the original function's name, docstring, and metadata in the wrapper function

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

Consider :

def decorator\_with\_args(msg):

def decorator(func):

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

print(msg)

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

return wrapper

return decorator

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

def login\_required(func):

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

if user.get("logged\_in"):

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

else:

print("Please log in")

return wrapper

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

class Circle:

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

self.\_radius = radius

@property

def area(self):

return 3.14 \* self.\_radius \*\* 2

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

def catch\_errors(func):

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

try:

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

except Exception as e:

print(f"Error: {e}")

return wrapper

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

Function decorators modify functions where as class decorators take a class and modify or add behavior.

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

Yes, but apply @classmethod or @staticmethod after the custom decorator.

3. Generators – 10 Questions

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

A function using yield to return values one at a time. It retains state and pauses or it resumes.

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

def even\_gen():

for i in range( 21):

if i % 2 == 0:

yield i

print(list(even\_gen()))

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

Raises StopIteration exception.

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

Yield is used in generators to return a value . It doesn't store all values in memory that can be said as generates on-the-fly.

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

A **generator expression** is similar to a list comprehension but uses **parentheses ()** instead of square brackets [].

Differnece :

* We use **list comprehension** when you **need all items immediately.**
* We use **generator expressions** for **large data sets or streaming,** where memory efficiency is important.

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

def nums():

for i in range(5):

yield i

print(nums())

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

def read\_file(file\_name):

with open(file\_name) as f:

for line in f:

yield line

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

The generator remembers the last yield and continues from there.

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

return ends the function.

yield pauses and continues from where it left off.

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

Converts all values from the generator to a list. Unlike normal functions, values are generated on-the-fly.

4. Iterators – 10 Questions

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

Iterable:

 An object that can be looped over (like lists, tuples, strings).

 It has the \_\_iter\_\_() method.

 It does **not** have a \_\_next\_\_() method.

 You can create an **iterator** from it using the iter() function.

 It can be used multiple times (reusable in loops).

Iterator:

 An object that represents a **stream of data.**

 It has both \_\_iter\_\_() and \_\_next\_\_() methods.

 Returns the next item using the next() function.

 Created from an iterable using iter(iterable).

 It is **not reusable** once exhausted (needs to be recreated).

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

class Count:

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

self.num = 0

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.num < 5:

self.num += 1

return self.num

else:

raise StopIteration

3. Explain what happens when StopIteration is raised.

When a StopIteration is raised , an iterator has no more items to return.

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

with open("file.txt") as f:

for line in iter(f.readline, ''):

print(line)

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

Calls iter() to get iterator and repeatedly calls next().

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

map(), filter(), zip(),enumerate() , iter() ,next()

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

it = iter([1, 2, 3])

while True:

try:

print(next(it))

except StopIteration:

break

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

class Squares:

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

self.num = 1

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.num <= 5:

val = self.num \*\* 2

self.num += 1

return val

else:

raise StopIteration

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

It does not reset. It repeated calls to next() raise StopIteration.

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

The itertools module in python with itertools is used for fast, memory-efficient tools like count(), cycle(), combinations(), permutations(), chain()

