Shallow Copy vs Deep Copy

1.What is a Shallow Copy?
A shallow copy creates a new object but copies only references of nested objects, so inner changes affect both objects.

2.What is a Deep Copy?
A deep copy creates a completely independent object by recursively copying all nested objects.

3.When do Shallow and Deep copies differ?
They differ when the object contains nested mutable objects like lists, dictionaries, or sets.

4.What modules/functions are used?
The copy module provides copy() for shallow copy and deepcopy() for deep copy.

5.Can simple assignments copy an object?
No, simple assignment only creates a new reference to the same object in memory.

6.How to test if a copy is deep or shallow?
Modify a nested object—if the original changes it's shallow, otherwise it's deep.

7.Are strings or integers affected?
No, because strings and integers are immutable.

8.Which is faster?
Shallow copy is faster than deep copy since it copies only references.

List vs Tuple

1.Are Lists and Tuples mutable?
Lists are mutable, while tuples are immutable.

2.Which is faster in performance?
Tuples are slightly faster due to immutability and memory optimization.

3.Can Lists and Tuples store different data types?
Yes, both are heterogeneous and can store mixed data types.

4.Do Tuples take less memory than Lists?
Yes, tuples use less memory because they are immutable and fixed in size.

5.When should you use a Tuple over a List?
Use a tuple when data should remain constant and safe from modification.

6.Can a Tuple be used as a dictionary key?
Yes, if all elements inside the tuple are immutable and hashable.

7.How do you convert between Lists and Tuples?
Use list() to convert to list and tuple() to convert to tuple.

8.Do both Lists and Tuples support indexing and slicing?
Yes, both support indexing and slicing, but only lists allow modification.

9.Can Tuples contain Lists?
Yes, tuples can contain lists, and the inner list can still be modified.

10.Which is better for function arguments or return values?
Tuples are better for fixed return values, while lists suit modifiable arguments.

List vs Set vs Dict Performance

1.What is the time complexity of membership testing in List, Set, and Dict?
List membership is O(n), while set and dict membership is O(1) on average.

2.Can Lists, Sets, and Dicts store duplicate values?
Lists allow duplicates, sets don't, and dict keys are unique but values can repeat.

3.Which data structures maintain insertion order?
Lists, tuples, and dictionaries maintain order, but sets do not.

4.Which structure is best for key-value mapping?
Dictionary is best due to fast O(1) average lookup and unique keys.

5.How do insertion and deletion operations compare?
Lists are slower (O(n)), while sets and dicts are faster (O(1) average).

6.How do you remove duplicates from a list efficiently?
Convert the list to a set and back to a list.

7.What happens when accessing a missing key in a dictionary?
It raises a KeyError.

8.Which data structures are mutable?
Lists, sets, and dicts are mutable; tuples and strings are immutable.

9.Which structure is best for counting frequency of items?
Dictionary is best for storing item counts with fast updates.

Python Libraries & Tools

1. The difference between @staticmethod and @classmethod?
@staticmethod doesn't access class or instance data, while @classmethod receives cls to work with class-level data.

2. The difference between generator and iterator?
An iterator implements __iter__() and __next__(), while a generator uses yield to automatically create an iterator.

3. What is decorator in python?
A decorator is a function that wraps another function to extend its behavior without modifying it.

4. Difference between list and tuple?
Lists are mutable and slower, while tuples are immutable, faster, and more memory efficient.

5. What are the primary built-in data structures/collections in python?
List, Tuple, Set, and Dictionary are the main built-in collections.

6. Exception Handling in python?
Exception handling uses try–except–else–finally to manage runtime errors safely.

OOP Concepts

7.What is Object-Oriented Programming (OOPs)?
OOP organizes code using classes and objects to improve reuse and structure.

8.Key OOP Principles in Python
Encapsulation binds data, abstraction hides complexity, inheritance reuses code, and polymorphism allows flexible behavior.

9.What is a Class and Object?
A class is a blueprint, and an object is its instance.

10.What is init() method?
It is a constructor that initializes object attributes during creation.

11.What is Inheritance?
Inheritance allows a child class to reuse parent class features.

12.What is Encapsulation?
Encapsulation protects data by bundling it with methods and controlling access.

13.What is Polymorphism?
Polymorphism allows the same interface to behave differently across objects.

14.What is Method Overriding?
It occurs when a child class redefines a parent class method.

15.What is super() in Python?
super() calls parent class methods inside a child class.

16.Difference between Class and Instance Variables
Class variables are shared by all objects, while instance variables are unique to each object.

Python Functions & Scope

1.How do you define a function in Python?
A function is defined using def with an optional return value.

2.What is the difference between a parameter and an argument?
A parameter is defined in the function; an argument is the value passed.

3.What are *args and **kwargs?
*args handles variable positional arguments and **kwargs handles keyword arguments.

4.What is the return statement used for?
It sends a value back and stops function execution.

5.What is recursion in Python?
Recursion is when a function calls itself with a base case.

6.What is a lambda function?
A lambda is a small anonymous one-line function.

7.What is scope in Python?
Scope defines variable visibility following the LEGB rule.

8.How do you access a global variable inside a function?
Use it directly or declare global to modify it.

9.What's the difference between local and global variables?
Local variables exist inside functions; global variables exist program-wide.

10.Can you return multiple values from a function?
Yes, Python returns them as a tuple.

11. Explain the code: Unique_email = f"{uuid.uuid4().hex[:8]}"
It generates a short unique 8-character hexadecimal string using a random UUID.

12. The difference between "is" and "=="?
== compares values, while is compares object identity.

✅ **Python & Test Automation – 50 Unique FAQ**

✅ **Python Basics**

1. What is Python?
Python is a high-level, interpreted programming language known for simplicity and readability, widely used in automation testing.

2. Why is Python popular in test automation?
Because of simple syntax, rich libraries (pytest, requests, selenium), fast development, and strong community support.

3. What are Python's key features?
Interpreted, object-oriented, dynamically typed, cross-platform, and supports multiple paradigms.

4. What is PEP 8?
PEP 8 is Python's coding style guide for writing clean and readable code.

5. What are Python data types?
int, float, string, list, tuple, set, dictionary, boolean.

6. What is indentation in Python?
Indentation defines code blocks instead of braces.

✅ Variables & Data Structures

7. Difference between list and tuple?
List is mutable; tuple is immutable and faster.

8. What is a dictionary?
A key-value pair collection used for fast data retrieval.

9. What is a set?
An unordered collection of unique elements.

10. Difference between list and set?
List allows duplicates; set doesn't.

11. Mutable vs immutable objects?
Mutable can change (list, dict); immutable cannot (string, tuple).

12. What is type casting?
Converting one data type to another (e.g., int(), str()).

✅ Control Flow

13. What are conditional statements?
if, elif, else used to control logic flow.

14. What loops are available in Python?
for loop and while loop.

15. What is break and continue?
break exits loop; continue skips iteration.

16. What is pass statement?
A placeholder that does nothing.

✅ Functions

17. What is a function?
A reusable block of code defined using def.

18. What are default arguments?
Function parameters with preset values.

19. What are *args and **kwargs?
*args = variable positional arguments; **kwargs = keyword arguments.

20. What is a lambda function?
Anonymous one-line function.

21. What is recursion?
A function calling itself.

✅ OOP Concepts

22. What is OOP?
Object-Oriented Programming using classes and objects.

23. What is a class and object?
Class = blueprint; object = instance.

24. What is inheritance?
Child class inherits parent properties.

25. What is encapsulation?
Restricting direct access to variables.

26. What is polymorphism?
Same method behaving differently.

✅ Exception & File Handling

27. What is exception handling?
Handling runtime errors using try-except.

28. What is try-except-finally?
try → risky code, except → error handling, finally → always runs.

29. Difference between error and exception?
Errors are syntax issues; exceptions occur during runtime.

30. How to read/write files in Python?
Using open(), read(), write(), close().

31. What are file modes?
r, w, a, r+, etc.

✅ Modules & Environment

32. What is a module?
A Python file containing reusable code.

33. What is pip?
Python package installer.

34. What is virtual environment?
Isolated Python environment for dependencies.

✅ Python for Test Automation

35. What is pytest?
A testing framework for writing simple and scalable test cases.

36. What are pytest fixtures?
Reusable setup/teardown functions.

37. What is parameterization in pytest?
Running a test with multiple data sets.

38. What is requests library?
Used for API testing and HTTP calls.

39. How do you validate API responses?
Status code, JSON schema, response body checks.

✅ Advanced Python Concepts

40. What is list comprehension?
Short syntax to create lists.

41. What is a generator?
Function using yield to produce values lazily.

42. Difference between deep copy and shallow copy?
Shallow copies reference objects; deep copies duplicate them.

43. What is a decorator?
Function that modifies another function.

44. What is multithreading?
Running multiple threads simultaneously.

45. What is GIL?
Global Interpreter Lock controls thread execution.

✅ Practical Testing & Framework Questions

46. How do you handle test data in Python?
Using CSV, JSON, Excel, or fixtures.

47. How do you log test execution?
Using Python logging module.

48. How do you connect Python with Jenkins?
Run pytest scripts via Jenkins pipeline.

49. How do you debug Python code?
Using print, logging, or debugger tools.

50. How do you optimize test automation scripts?
Reusable functions, modular design, fixtures, parallel execution.

✅ Shallow Copy vs Deep Copy — Code with Output

```
import copy


original = [[1, 2], [3, 4]]


shallow = copy.copy(original)

shallow[0][0] = 99
```

```
print("Original after shallow:", original)
print("Shallow:", shallow)

deep = copy.deepcopy(original)
deep[0][0] = 100

print("Original after deep:", original)
print("Deep:", deep)
```

Output:

```
Original after shallow: [[99, 2], [3, 4]]
Shallow: [[99, 2], [3, 4]]
Original after deep: [[99, 2], [3, 4]]
Deep: [[100, 2], [3, 4]]
```

✅ List vs Tuple — Code with Output

```
my_list = [1, 2, 3]
my_list[0] = 10
print("List:", my_list)

my_tuple = (1, 2, 3)
print("Tuple:", my_tuple)
```

Output:

```
List: [10, 2, 3]
Tuple: (1, 2, 3)
```

✅ List vs Set vs Dict — Code with Output

```
lst = [1, 2, 2, 3]
st = {1, 2, 2, 3}
```

```
dct = {"a": 1, "b": 2}


print("List:", lst)
print("Set:", st)
print(2 in lst)
print(2 in st)
print("a" in dct)
```

Output:

```
List: [1, 2, 2, 3]
Set: {1, 2, 3}
True
True
True
```

✅ @staticmethod vs @classmethod — Code with Output

```
class Example:
    count = 0

    @staticmethod
    def static_method():
        return "No class access"

    @classmethod
    def class_method(cls):
        cls.count += 1
        return cls.count

print(Example.static_method())
print(Example.class_method())
```

Output:

No class access

1

✅ **Generator vs Iterator — Code with Output**

```
def gen():
    for i in range(3):
        yield i

g = gen()
print(next(g))
print(next(g))

lst = [1, 2, 3]
it = iter(lst)
print(next(it))
```

Output:

0

1

1

✅ **Decorator — Code with Output**

```
def decorator(func):
    def wrapper():
        print("Before function")
        func()
        print("After function")
    return wrapper
```

```
@decorator
def say_hello():
    print("Hello!")

say_hello()
```

Output:

```
Before function
Hello!
After function
```

✅ OOP (Class & Object) — Code with Output

```
class Person:
    def __init__(self, name):
        self.name = name

    def greet(self):
        print("Hello,", self.name)

p = Person("Sam")
p.greet()
```

Output:

```
Hello, Sam
```

✅ Inheritance + super() — Code with Output

```
class Parent:
    def show(self):
        print("Parent method")

class Child(Parent):
```

```
    def show(self):
        super().show()
        print("Child method")


c = Child()
c.show()
```

Output:

```
Parent method
Child method
```

✅ *args / **kwargs — Code with Output

```
def demo(*args, **kwargs):
    print("Args:", args)
    print("Kwargs:", kwargs)


demo(1, 2, name="Sam", age=25)
```

Output:

```
Args: (1, 2)
Kwargs: {'name': 'Sam', 'age': 25}
```

✅ Lambda Function — Code with Output

```
square = lambda x: x * x
print(square(5))
```

Output:

```
25
```

✅ Recursion — Code with Output

```
def factorial(n):
```

```
    if n == 1:
        return 1
    return n * factorial(n - 1)

print(factorial(5))
```

Output:

```
120
```

✅ Exception Handling — Code with Output

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
finally:
    print("Done")
```

Output:

```
Cannot divide by zero
Done
```

✅ File Handling — Code with Output

```
with open("test.txt", "w") as f:
    f.write("Hello Python")

with open("test.txt", "r") as f:
    print(f.read())
```

Output:

```
Hello Python
```

✅ Pytest — Simple Test Example

```
def add(a, b):
    return a + b


def test_add():
    assert add(2, 3) == 5
```

Output (pytest run):

```
================ test session starts ================
collected 1 item


test_file.py .                          [100%]


================ 1 passed ======================
```
