Python

Shallow Copy vs Deep Copy

1.What is a Shallow Copy?

A **shallow copy** creates a new object, but it **copies only the references** of nested objects, not the actual nested data.
So, changes in nested (inner) objects will reflect in both the original and the copied object.

2.What is a Deep Copy?

A **deep copy** creates a new object and **recursively copies all nested objects**, so the original and copied objects are completely independent.

3.When do Shallow and Deep copies differ?

Shallow and deep copies differ **when the object contains nested (mutable) objects** like lists, dictionaries, or sets.

4.What modules/functions are used?

Python uses the **copy module** for both shallow and deep copying.

5.Can simple assignments copy an object?

No, **simple assignment does not create a copy** of an object.
It only creates a **new reference to the same object in memory**.

6.How to test if a copy is deep or shallow?

You can test whether a copy is deep or shallow by **modifying a nested object** and checking if the original changes.

7.Are strings or integers affected?

No, **strings and integers are not affected** by shallow or deep copy differences because they are **immutable**.

8.Which is faster?

Shallow copy is faster because it copies only references, while deep copy recursively copies all objects.

List vs Tuple

1.Are Lists and Tuples mutable?

Lists are mutable, but tuples are immutable.

2.Which is faster in performance?

Tuples are slightly faster than lists in Python.

Reason:

- Tuples are **immutable**, so Python can optimize memory and access.
- Lists are **mutable**, so they have extra overhead for resizing and modifications.

3.Can Lists and Tuples store different data types?

One-liner:

Yes, both lists and tuples can store multiple data types.

Explanation:
Python lists and tuples are **heterogeneous**, meaning they can hold elements of different types in the same collection.

4.Do Tuples take less memory than Lists?

One-liner:

Yes, tuples generally take **less memory** than lists.

Reason:

- Tuples are **immutable**, so they don't need extra space for resizing or modification.
- Lists are **dynamic**, so they allocate extra memory to support growth.

5.When should you use a Tuple over a List?

Use a tuple when the data should not change and you want better performance and safety.

6.Can a Tuple be used as a dictionary key?

One-liner:

Yes, a tuple can be used as a dictionary key **if it contains only immutable elements**.

Why?

- Dictionary keys must be **hashable**.
- Tuples are hashable **only when all their elements are immutable**.

7.How do you convert between Lists and Tuples?

You can convert a tuple to a list using list(tuple_obj) and a list to a tuple using tuple(list_obj).
This is useful to switch between **mutable (list)** and **immutable (tuple)** depending on whether you need to modify the data or keep it fixed.

8.Do both Lists and Tuples support indexing and slicing?

Python

Yes, both lists and tuples support **indexing and slicing**.
You can access elements using indexes (obj[0]) and extract sub-parts using slicing (obj[1:4]), but only lists allow modification of values.

9.Can Tuples contain Lists?

Yes, tuples can contain lists.
The tuple itself is immutable, but the list inside it is mutable, so the inner list can be modified even though the tuple cannot be reassigned.

10.Which is better for function arguments or return values?

Tuples are better for function return values when the data should not be modified, as they ensure immutability and safety.
Lists are better for function arguments when the data needs to be changed or updated inside the function.

List vs Set vs Dict Performance

1.What is the time complexity of membership testing in List, Set, and Dict?*

One-liner:

Membership testing (in) is **O(n)** for lists and **O(1)** for sets and dictionaries.

Explanation:

- **List:** Checks elements one by one → **Linear search** → **O(n)**
- **Set:** Uses hash table → **Average O(1)**
- **Dict:** Checks keys using hash table → **Average O(1)**

2.Can Lists, Sets, and Dicts store duplicate values?

One-liner:

Lists allow duplicates, sets do not, and dictionaries allow duplicate values but not duplicate keys.

Explanation:

- **List:** Can store the same value multiple times.
- **Set:** Automatically removes duplicates.
- **Dict:** Keys must be unique, but values can be duplicated.

3.Which data structures maintain insertion order?

Lists and tuples always maintain insertion order; dictionaries also maintain insertion order (from Python 3.7+), but sets do not.

4.Which structure is best for key-value mapping?

Python

One-liner:

Dictionary (dict) is the best data structure for key–value mapping.

Reason:

- Designed specifically for **key–value pairs**.
- Provides **fast lookup, insertion, and deletion (O(1) average)**.
- Keys are **unique and hashable**.

5.How do insertion and deletion operations compare?

One-liner:

Lists are slower for insertion/deletion, while sets and dictionaries are faster.

Explanation:

- **List:** Insertion/deletion in the middle → **O(n)** (elements must be shifted).
- **Set:** Insertion/deletion → **O(1)** average.
- **Dict:** Insertion/deletion → **O(1)** average.

6.How do you remove duplicates from a list efficiently?

One-liner:

Convert the list to a set and back to a list.

Example:

```
lst = [1, 2, 2, 3, 3, 4]
unique = list(set(lst))
```

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

One-liner:

Accessing a missing key using dict[key] raises a **KeyError**.

Example:

```
d = {"a": 1}
print(d["b"])   # ✕ KeyError
```

8.Which data structures are mutable?

One-liner:

Lists, sets, and dictionaries are **mutable**, while tuples and strings are **immutable**.

Explanation:

Python

- **Mutable:** list, set, dict → can be changed after creation.
- **Immutable:** tuple, str, int, float → cannot be changed.

9.Which structure is best for counting frequency of items?

One-liner:

Dictionary is best for counting frequencies.

Reason:

- Stores item as **key** and count as **value**.
- Provides **O(1)** average update and lookup.

Python Libraries & Tools:

---

1. The difference between @staticmethod and @classmethod?

@staticmethod is used when the method does not need access to the class or instance; it behaves like a normal function placed inside a class for logical grouping. @classmethod receives the class as its first argument (cls) and is used when you need to work with or modify class-level data, often for creating alternative constructors.

2. The difference between generator and iterator?

An **iterator** is an object that implements the __iter__() and __next__() methods and returns values one at a time.
A **generator** is a simpler way to create an iterator using a function with the yield keyword; it automatically handles iteration and state without writing those methods.

3. What is decorator in python?

A **decorator** in Python is a function that **modifies the behavior of another function or method without changing its source code**.
It is used to add extra functionality (like logging, authentication, timing) by wrapping the original function using the @decorator_name syntax.

4. Difference between list and tuple?

The main difference between a **list** and a **tuple** is that lists are **mutable** (can be changed), while tuples are **immutable** (cannot be changed).
Lists use more memory and are slower, whereas tuples are more memory-efficient and slightly faster, making them suitable for fixed data.

5. What are the primary built-in data structures/collections in python?

The primary built-in data structures in Python are **List, Tuple, Set, and Dictionary**. They are used to store collections of data, where lists are ordered and mutable,

tuples are ordered and immutable, sets store unique values, and dictionaries store key–value pairs.

6. Exception Handling in python?

Exception handling in Python is used to **handle runtime errors gracefully** without crashing the program.
It is done using try, except, else, and finally blocks, where try contains risky code and except handles the error.

7.What is Object-Oriented Programming (OOPs)?

Object-Oriented Programming (OOPs) is a way of writing programs using **objects and classes**, where data and functions are grouped together.
It helps make code **more organized, reusable, and easier to maintain**.

8.Key OOP Principles in Python

- Encapsulation – Bind data and methods in a class.
- Abstraction – Hide complex implementation details.
- Inheritance – Reuse code from parent classes.
- Polymorphism – Same method name, different behavior.

9.What is a Class and Object?

A **class** is a blueprint or template used to create objects.
An **object** is an instance of a class that represents a real-world entity and contains data and behavior defined by the class.

10.What is __init__() method?

The __init__() method is a special method in Python that acts as a **constructor**.
It is automatically called when an object is created and is used to **initialize (set up) the object's data**.

11.What is Inheritance?

Inheritance is an OOP concept where one class (child class) **inherits properties and methods from another class (parent class)**.
It helps in **code reusability** and allows creating new classes based on existing ones.

12.What is Encapsulation?

Encapsulation is an OOP concept that **bundles data and methods into a single unit (class)** and restricts direct access to some data.
It helps in **data protection and controlled access** using methods and access modifiers.

13.What is Polymorphism?

Polymorphism means **one interface, many forms**.
It allows the same function or method name to behave differently for different objects, improving flexibility and code reuse.

14.What is Method Overriding?

Method overriding is when a **child class provides its own implementation of a method that is already defined in the parent class**.
It allows the child class to change or customize the behavior of the inherited method.

15.What is super() in Python?

super() is a built-in function used to **call methods or access attributes from the parent class**.
It is commonly used in inheritance to extend or reuse the parent class functionality without rewriting code.

16.Difference between Class and Instance Variables

Variable Type	Description
Class Variable	Shared by all objects of the class
Instance Variable	Unique for each object

Python Functions & Scope:

1.How do you define a function in Python?

A function in Python is defined using the def keyword followed by the function name and parentheses.
The function body is written inside an indented block and can return a value using the return statement.

2.What is the difference between a parameter and an argument?

A **parameter** is the variable defined in the function definition.
An **argument** is the actual value passed to the function when it is called.

3.What are `*args` and `**kwargs`?

*args is used to pass a **variable number of positional arguments** to a function.
kwargs is used to pass a **variable number of keyword arguments (key–value pairs) to a function.

4.What is the return statement used for?

Python

The return statement is used to **send a value back from a function to the caller**.
It also **ends the execution of the function** at that point.

5.What is recursion in Python?

Recursion is a technique where a **function calls itself** to solve a problem.
It works by breaking the problem into smaller parts and must have a **base case** to stop the recursion.

6.What is a lambda function?

A **lambda function** is a small, **anonymous (nameless) function** defined using the lambda keyword.
It is used for short, one-line operations and can take any number of arguments but only one expression.

7.What is scope in Python?

Scope in Python refers to the **region where a variable is accessible** in the program.
Python follows the **LEGB rule**: Local, Enclosing, Global, and Built-in scopes.

8.How do you access a global variable inside a function?

You can access a global variable inside a function by using it directly by its name.
If you want to **modify** the global variable inside the function, you must use the global keyword.

9.What's the difference between local and global variables?

A **local variable** is defined inside a function and is accessible only within that function.
A **global variable** is defined outside all functions and can be accessed throughout the program.

10.Can you return multiple values from a function?

Yes, a function can return multiple values by returning them as a **tuple**.
Python automatically packs the values into a tuple and unpacks them when assigned.

11. Explain the code: Unique_email = f"{uuid.uuid4().hex[:8]}"

This code generates a **random unique string**.
uuid.uuid4() creates a random UUID, .hex converts it to a hexadecimal string, and [:8] extracts the first 8 characters to create a short unique value, often used for unique IDs or emails.

12. The difference between "is" and "=="?

Python

== is used to **compare the values** of two objects.
is is used to **check if both variables refer to the same object in memory** (same identity).