Mutable vs Immutable in Python

In Python, objects are classified into two main categories: **mutable** and **immutable**. Understanding this difference is crucial because it affects how variables behave in memory and how data is shared or modified.

Immutable Objects

Immutable objects cannot be changed after creation. Any modification creates a new object in memory.

1. int
2. float
3. bool
4. str
5. tuple
6. frozenset

Example:

```
x = 10
y = x
x += 5
```

Here, x and y refer to different objects because integers are immutable.

Mutable Objects

Mutable objects can be changed after creation. Modifying them updates the same object in memory.

1. list
2. dict
3. set
4. bytearray
5. custom class objects (by default)

Example:

```
a = [1, 2, 3]
b = a
a.append(4)
```

Both a and b now reference the same list object because lists are mutable.

Mutable vs Immutable Summary

1. Immutable: cannot be changed; any modification creates a new object.
2. Mutable: can be modified in place without changing the object's identity.
3. Class instances are mutable by default (attributes can be changed).

4 Use immutability to ensure data consistency and thread-safety.

Quick Reference Table:

Immutable: int, str, bool, tuple, frozenset
Mutable: list, dict, set, class objects