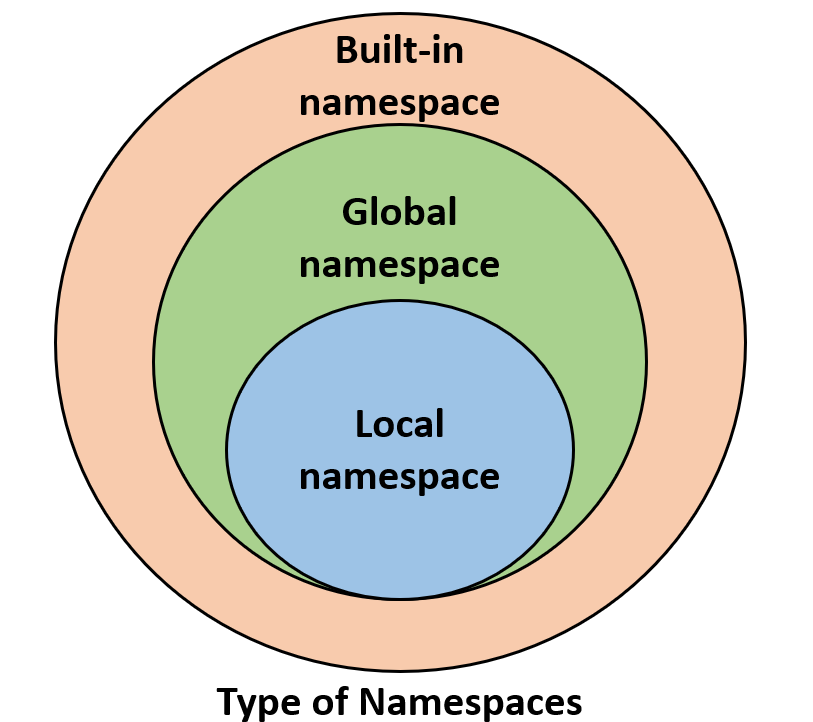
**Python Namespace and Scope**

In Python, a namespace is a system that provides a way to organize and control the visibility of names (identifiers) in a program. Namespaces are essential for avoiding naming conflicts and managing the scope of variables and objects. Understanding namespaces and scope is crucial for writing clean and maintainable code in Python.

**1. Namespace:**

* A namespace is a mapping from names to objects. It ensures that names are unique and can be used without conflict within a particular scope.
* Python implements namespaces using dictionaries, where the keys are the names (identifiers) and the values are the corresponding objects.
* Namespaces in Python can be classified into three main categories: built-in namespaces, global namespaces, and local namespaces.

**Types of Namespaces:**

****

There are mainly three types of namespaces in Python:

* **Built-in Namespace:** This namespace contains built-in functions like print() and id() that are always available.
* **Global Namespace:** When you create a module, a global namespace is created, which encompasses all the functions, variables, etc., defined at the global level.
* **Local Namespace:** Local functions or methods create their own local namespaces. These namespaces are limited to the scope of the function or method where they are defined.

**Lifetime of a Namespace:**

The lifetime of a namespace depends on the scope of objects. Once the scope of an object ends, the lifetime of that namespace also ends. This means that objects in inner namespaces cannot be accessed from outer namespaces.

**2. Scope:**

* Scope refers to the visibility and accessibility of names within a program. It determines where a variable or object can be referenced or modified.
* **Python has four levels of scope:**
  + **built-in scope**
  + **global scope**
  + **enclosing (non-local) scope\**
  + **local scope**
* The scope of a name depends on where it is defined in the code and follows the LEGB rule: Local, Enclosing, Global, Built-in.

**3. Examples:**

* **Global Scope:** Variables defined outside of any function or class have global scope and can be accessed from anywhere in the program.

```python

x = 10 # Global variable

def func():

print(x) # Accessing global variable

func() # Output: 10

```

* **Local Scope:** Variables defined within a function have local scope and can only be accessed within that function.

```python

def func():

y = 20 # Local variable

print(y) # Accessing local variable

func() # Output: 20

```

* **Enclosing (Non-local) Scope:** If a function is defined within another function, it can access variables from the outer function's scope.

```python

def outer():

z = 30 # Enclosing variable

def inner():

print(z) # Accessing enclosing variable

inner()

outer() # Output: 30

```

* **Built-in Scope:** It includes the names of built-in functions and modules provided by Python (e.g., `print()`, `len()`, `range()`).

**4. Namespace Pollution/ Mistakes:**

Namespace pollution occurs when the same name is used to refer to different objects within the same scope, leading to confusion and errors.

It is essential to use meaningful and unique names to avoid namespace pollution and maintain code readability.

Understanding namespaces and scope is crucial for writing efficient and organized Python code. By managing namespaces effectively, developers can prevent naming conflicts, improve code maintainability, and enhance overall program performance.